NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

IMPLEMENTATION OF A NATURAL LANGUAGE PROCESSOR USING FUNCTIONAL GRAMMAR

by

Fred G. Orchard

December 1985

Thesis Advisor: Roger G. Marshall

CURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
. DECLASSIFICATION / DOWNGRADING SCHEDULE	
PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
. NAME OF PERFORMING ORGANIZATION Naval Postgraduate Schoo 6b. OFFICE SYMBOL (If applicable) 52	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5100	7b. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5100
. NAME OF FUNDING / SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

. TITLE (Include Security Classification)
IMPLEMENTATION OF A NATURAL LANGUAGE PROCESSOR USING FUNCTIONAL GRAMMAR

. PERSONAL AUTHOR(S)
Orchard, Fred G.

a TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Master's Thesis	FROM _____ TO _____	1985 December	81

. SUPPLEMENTARY NOTATION

COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Functional Grammar, Natural Language Processor, Pragmatics, Predications

ABSTRACT (Continue on reverse if necessary and identify by block number)

This thesis presents the design and implementation of a natural language processor using Functional Grammar. Traditionally, grammars have consisted of a set of words and a set of semantic and syntactic rules which combine the words to form sentences. Thus, the language is looked at as a syntactic structure which is used to derive meaning. Functional Grammar looks at language as a means of social interaction and applies the syntactic and semantic rules only after the meaning, based on pragmatics, of the sentence has been established. Prolog has been used to demonstrate how Functional Grammar can be used to provide that meaning.

). DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
a NAME OF RESPONSIBLE INDIVIDUAL Prof. MacLennan	22b. TELEPHONE (Include Area Code) (408) 646-2509 22c. OFFICE SYMBOL Code 52Ml

D FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

Approved for public release; distribution is unlimited.

Implementation of a Natural Language Processor Using Functional Grammar

by

Fred Gregg Orchard
Lieutenant Commander, United States Navy
B.S., United States Naval Academy, 1973

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1985

ABSTRACT

This thesis presents the design and implementation of a natural language processor using Functional Grammar. Traditionally, grammars have consisted of a set of words and a set of semantic and syntactic rules which combine the words to form sentences. Thus, the language is looked at as a syntactic structure which is used to derive meaning. Functional Grammar looks at language as a means of social interaction and applies the syntactic and semantic rules only after the meaning, based on pragmatics, of the sentence has been established. Prolog has been used to demonstrate how Functional Grammar can be used to provide that meaning.

TABLE OF CONTENTS

ACKNOWLEDGMENT

This thesis was written under the direction of Professor Roger G. Marshall. His exceptional knowledge of Functional Grammar in particular, and of Computer Science in general, was an inspiration and an invaluable asset in the completion of this project.

I. INTRODUCTION

Many attempts have been made to program computers to understand natural language. Terry Winograd's "Programmer" project provides an excellent example. [Ref. 1:pp. 80-108] Natural language processing is necessarily based on grammars that have been developed by linguists. Although many grammars have been developed, Noam Chomsky's Transformational Grammar has been in the forefront of linguistic study for the past 30 years. Recently, Simon C. Dik proposed a new grammar, based on a functional paradigm. Most grammars are based on the idea that languages are a set of sentences. Dik's Functional Grammar (FG) is based on the concept that language is a means of social interaction. His ideas mark a radical departure from the current theories. [Ref. 2:p. 1] This thesis will attempt to provide a working model of FG utilizing Prolog as the implementation language. The program will evaluate a paragraph of text and return the theme of the paragraph. It will also determine whether the paragraph is consistent in its theme.

To understand FG, a discussion of some major grammar theories is needed. A discussion of Traditional Grammar, Transformational Grammar, Case Grammar, and Conceptual Analysis will trace the history of grammar theory in the twentieth century and provide the background necessary to understand FG.

A. TRADITIONAL GRAMMAR

Traditional Grammar is the grammar that most laymen recognize. It is the grammar borne out of a necessity to educate millions of youngsters in a formal manner. Traditional Grammar is based on a set of definitions and prescriptive rules. The definitions are those such as:

* A NOUN is a person, place, or thing.
* A SENTENCE expresses a complete thought.

Prescriptive rules are those such as:

* Never split infinitives.
* Don't end a sentence with a preposition.

The grammar is taught by counterexample. That is, a student is presented with sentences which he must make "right" by application of the given rules. It develops in one an intuitive understanding of the language, but does not give the user an explicit algorithm for constructing sentences.

B. TRANSFORMATIONAL GRAMMAR

The break from the traditionalists came in the 1950's. Linguists were generally divided into two groups. The first group, known as the structuralists, believed that languages were derived separately and that any commonality between languages was purely coincidental. The other group explored the possibility that all languages came from a single source or maybe only a few sources. This theory would explain similarities between languages and suggest a possible vehicle for evaluating all languages in the same manner.

Among this latter group was Noam Chomsky. In 1957, he published Syntactic Structures. [Ref. 3] In this, he developed his theory of Generative-Transformational Grammar or as it is more commonly known, Transformational Grammar (TG). In 1965, he refined and modified his theory in Aspects of Syntactic Theory. [Ref. 4] This book was destined to become the yardstick by which all new grammars were measured.

Transformational Grammar gives explicit rules for processing sentences. These rules are of two types (1) Phrase Structure Rules (PSR) and (2) Transformational Rules (TR). The "surface structure" of a sentence is the readable form, that is, the way it appears in print. PSRs provide a path from the sentence's surface structure to its "deep structure". The deep structure provides a description of the syntactic functions that each word performs in the sentence, and with the exception of SI, is in a one-to-one correspondence with the surface structure. SI is an indicator of the type of sentence. From this deep structure, TR's are used to transform the sentence back into surface structure and into other deep structures with similar meanings. An example will clarify this process. The following is a set of PSR's:

(1) Sent -> SI + NP + Aux + VP
(2) SI -> (pos)|(neg)|(com)|(quest)|(pass)
(3) VP -> V + NP|PP|Adj|e
(4) PP -> Prep + NP
(5) NP -> (Article|e) + N + (Sent|e)
(6) Aux -> Tense Marker + (Modal|e)
(7) Modal -> (can)|(may)|(shall)|(will)

SI=Sentence Indicator, NP= Noun Phrase, VP= Verb Phrase,
PP= Prepositional Phrase, e= empty

(1) The man went to the store.

Sentence (1) will be transformed from the surface structure to the deep structure by use of the above PSR's. Brackets are used for clarification. Sentence (1) is diagramed in Figure 1.1[Ref. 5:pp. 78-80]

STEP	RULE
a. Sent-)SI + NP Aux + VP	1
b. Sent-)[positive]+ NP + Aux + VP	2
c. Sent-)[positive]+[(Article)+N+(Sent)]+Aux+VP	5
d. Sent-)[positive]+[the+man]+Aux+VP	
e. Sent-)[positive]+[the+man]+[Tense Marker+(Modal)]+VP	6
f. Sent-)[positive]+[the+man]+[past]+VP	
g. Sent-)[positive]+[the+man]+[past]+[V+PP]	3
h. Sent-)[positive]+[the+man]+[past]+[go+[Prep+NP]]	4
i. Sent-)[positive]+[the+man]+[past]+[go+[to+[Article+N]]]	5
j. Sent-)[positive]+[the+man]+[past]+[go+[to+[the+store]]]	

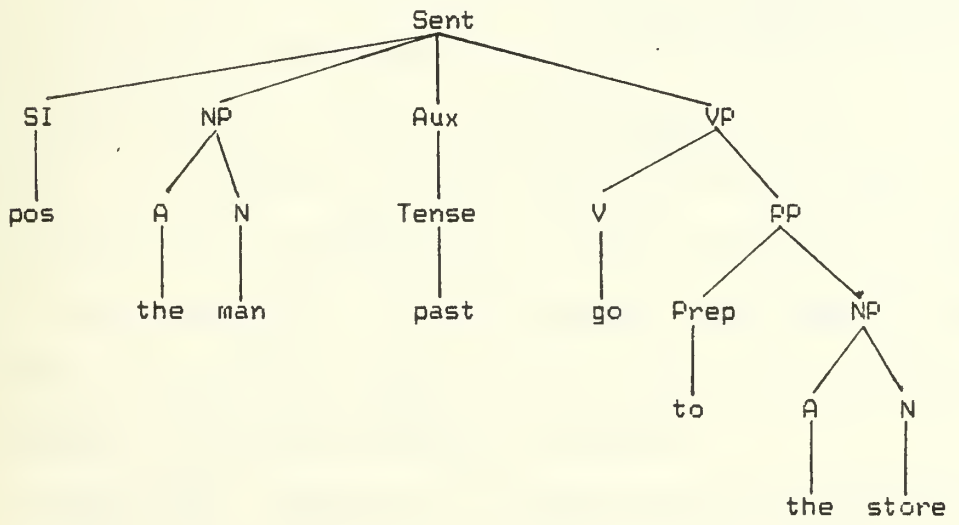

Figure 1.1 Sentence (1) Diagram

Transformational Rules provide a means of changing the form of the sentence. There are many types of TR's. Some TR's, such as the question transform, the negation transform, and the command transform, change the meaning as well as the structure. For example, applying the question transform to the sentence above results in the surface structure "Did the man go to the store?". To make the transformation requires two steps. First, the deep structure of the sentence is changed to reflect the new form. Then the new surface structure is derived from the new deep structure. Some TR's do not change the meaning, yet they must also have their deep structure changed in order to arrive at the new surface structure. This requirement to change the deep structure of a sentence in order to transform it is one of the deficiencies of TG. The deep structure should express the meaning of the sentence, but in the case of TG, it must be changed for different configurations of the same sentence. Consider the following sentences:

(2) John read the book.

(3) The book was read by John.

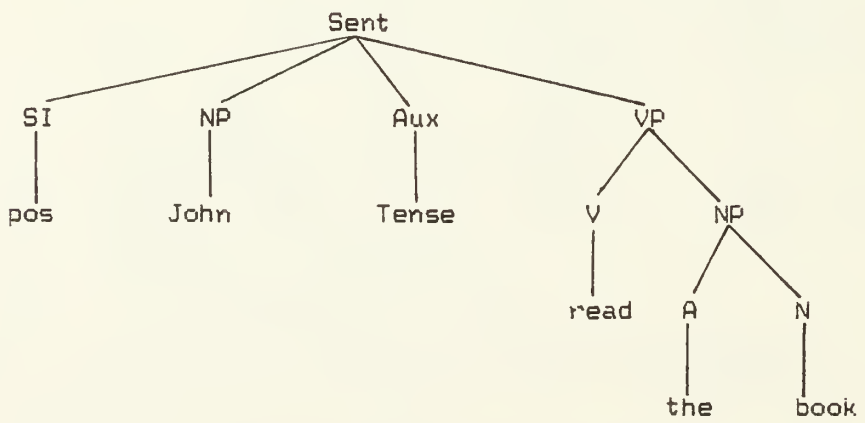

Figure 1.2 Sentence (1) diagram

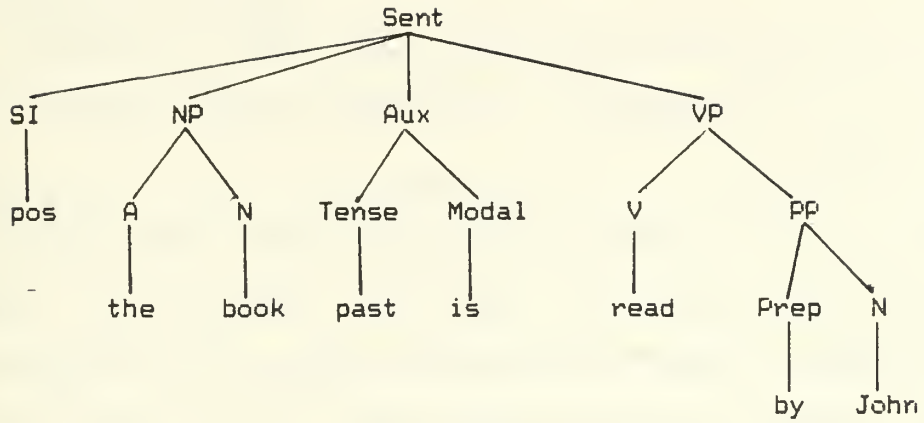

Figure 1.3 Sentence (2) diagram

Figures 1.2 and 1.3 show the diagrams of sentences ((2) and (3), respectively. While it is true that syntactically, these sentences are different, semantically they are the same. "There exists a book which was read by John." As will be seen later, Functional Grammar treats this situation in a completely different manner.[Ref. 5:pp. 81-88]

C. CASE GRAMMAR

When Chomsky published his Aspects of the Theory of Syntax, many linguists questioned the primacy of syntax. Among these was Charles Fillmore. Fillmore subsequently published "A case for Case" where he presented his new Case Grammar (CG).[Ref. 6] Fillmore maintained that the semantics of a sentence should dictate the syntax of the sentence. To illustrate this, he developed a set of semantic cases. These cases, although not exhaustive, are considered the minimum necessary to adequately process a language.

* Agentive (A)- the person or animate object that performs the action specified by the verb.

* Instrumental (I)- the object or force used by the verb.

* Dative(D)- the person or animate object affected by the verb.

* Factitive (F)- the object or being resulting from the action of the verb.

* Locative (L)- the location or orientation specified by the verb.

* Objective (O)- things that are affected by the action of the verb.

A sentence is composed of two parts, modality and proposition (Sent -> M+P). Proposition is further decomposed into a verb and a number of cases, results in the following construction: Sent -> M + V + C1 + C2 + ...+ Cn. The cases are from the above list. A sample sentence might be decomposed as follows:

* John gave the book to Mary.

a. Sent -> M + V + C1 + C2 +...+ Cn

b. Sent -> past + V + C1 + C2 +...+ Cn

c. Sent -> past + give + C1(A) + C2(O) + C3(D)

d. Sent -> past + give + John + book + Mary

John is the Agentive case, book is the Objective case, and Mary is the Dative case. By this process, Fillmore was able to capture some of the semantics of a sentence. But CG was still a Transformational Grammar requiring transforming of the deep structure to change a sentence into a similar sentence with the same meaning. Additionally, Fillmore's cases were hardly exhaustive and did not capture the context of a group of sentences. [Ref. 6:pp. 21-31]

D. CONCEPTIONAL ANALYSIS

At about the same time that Fillmore was developing his Case Grammar, Roger Schank presented a new approach to natural language processing. He felt that transformations of structures and syntactic parsing were not the right direction for natural language processing. Instead, he offered his theory of Conceptional Analysis of Language(CAL). CAL deals with the meaning of the sentence rather than the structure and the syntax. Thus sentences such as (4) and (5), below, have the same meaning even though their structures are quite different.

(4) John gave the book to Mary.

(5) The book was given to Mary by John.

Schank developed a set of conceptual cases. These cases, which were language independent, would capture the conceptual content of a word or phrase. A sentence was evaluated and each time a word or phrase was encountered which had the meaning of a particular conceptual case, that case was substituted for the word or phrase. At the end of the sentence evaluation, one meaning was achieved. A program using this approach was developed at Stanford University.[Ref. 7:pp. 187-247]

E. FUNCTIONAL GRAMMAR

Shortly after Transformational Grammar and Case Grammar gained respectability, Simon Dik published a dissertation challenging Chomsky's Transformational Grammar. His main criticisms were that the treatment of syntax and semantics was inadequate and that a non-transformational method was possible. He formalized his theory of

Functional Grammar in 1978 with the publication of Functional Grammar. Dik based his theory on Functional Grammar on two principles:

(1) A language system should be more than just a grammar that conforms to a set of syntactic rules. It must also be able to explain the ultimate use of the rules and how they are interpreted.

(2) A language system should be devised so that it can most easily be incorporated into a wider pragmatic theory of verbal interaction. [Ref. 2:p. 2]

Dik's theory of Functional Grammar differs from Transformational Grammar in some very basic ways. Transformational Grammars are defined by a set of syntactic rules. These rules state that a noun phrase must be followed by a verb phrase, or a noun phrase may be preceeded by an adjective, etc. Because of this dependence on the syntax of the language, the language is forced into a priority system where syntax comes first, followed by semantics, and lastly by pragmatics. On the other hand, Dik's Functional grammar has the language being defined as a method of social interaction. Starting from this premise, Dik ends up with a priority system that places pragmatics at the top of the list, followed by semantics, and lastly by syntax.

A Functional Grammar should be able to adequately recognize all of the linguistic expressions of a particular language. This requires defining rules which cover what Dik terms "the most significant generalizations of the language."[Ref. 2:p 6] Standards of adequacy are used in grammar development as a baseline assessment tool. After exploring the many standards available, Dik established three standards of adequacy that he felt FG had to conform to in order to be usable.

The first of these is pragmatic adequacy. This is the heart of Functional Grammar theory. A Functional Grammar must recognize the properties of lingustic expressions from the everyday use of the language. Dik believes that FG needs to resolve the pragmatic aspect before any resolution of the semantic and syntactic functions of a linguistic expression can be attempted. If such is the case, then it shall be considered pragmatically adequate.

The second standard is that of psychological adequacy. To be considered psychologically adequate, a Functional Grammar must be consistent with strongly established and supported psychological hypotheses about language. An example of this type of concern would be:

Q: How did you arrive?

A: By United.

Without knowledge of airline company names, one might not understand the answer to this question. This is a general expression accepted in our language yet not served by any syntactic, semantic, or pragmatic rules. It is therefore contained in what Dik considers psychological issues.

The third and last standard is that of typological adequacy. A Functional grammar must be applicable to typologically different languages while at the same time addressing the similarities and differences of the languages. A Functional Grammar that accomplishes this would be considered typologically adequate. [Ref. 2: pp 6-9]

A grammar can fail in two ways. It can be so constrained that it does not include expressions that assure that it attains descriptive adequacy. That is, it does not fully describe the expressions that the language requires. On the other hand, it could be so large that too much is included. Functional Grammar's approach to this is to restrict the range of descriptive devices allowed. This is accomplished in three ways.

Functional Grammar uses very few transformations. Transformations are of two types, those that effect changes in pre-established structures and "structure sensitive" transformations. The latter are rules in which the elements are affected by the environment that the rule is in but do not alter the structures. Functional Grammar does not allow transformations of the first type with the one exception that it allows for deletion of variables in certain situations. Essentially, no other structure changes are allowed.

Functional Grammar uses no filtering devices, which are used extensively in transformational grammars. A set of expressions is expanded into a larger superset of expressions which are used to evaluate sentences that are not in the orthodox form of (noun phrase)(verb phrase). After evaluation of the sentence, the useless structures of the set are "filtered out." This allows a lot of freedom but adds extra structures that later must be discarded. Functional Grammars strive to immediately recognize the target set of well-formed expressions thereby negating the need for filtering devices.

Functional Grammar also differs in its treatment of lexical items, The lexical items are the basic words, punctuation, and their usage.

The abstraction of a grammar is constrained by its definition of lexical items. As a grammar gains more and more lexical items, the possible combinations grow exponentially. A grammar must deal with all of these combinations. Most grammars do so by constraining the number of lexical items, which in turn constrains the grammar. Because Functional Grammar puts the treatment of lexical items at the bottom of its priority list rather than the top it is less constrained than traditional grammars. [Ref. 2:pp 10-12]

Functional Grammar is a radical approach to linguistic theory when looked at from the Chomsky point of view. However, it compares favorably with the traditional approach. The traditional approach allows one to develop an intuitive understanding of the grammar but it does not provide an algorithm to build the language. Functional Grammar maintains that intuitive understanding, Dik's "method of social interaction," and provides the algorithm missing in Traditional Grammar.

II. FUNCTIONAL GRAMMAR

In Functional Grammar, language is defined first to be an instrument of social interaction, which is opposed to Transformational Grammar's view of language as being a set of sentences. To provide a framework in which the FG definition will work, Functional Grammar utilizes the following definitions:

* Predication - an expression that governs the application of a predicate to an appropriate number of terms functioning as arguments of that predicate.
* Constituent - a term acting as an argument of a predicate.
* Syntactic function - the role a constituent plays in presenting the perspective from which the state of affairs is presented in the linguistic expression.
* Semantic function - the meaning a constituent has within the state of affairs presented in the predication.
* Pragmatic function - the informational status of the constituent in the context that the predication exists. [Ref. 2:p. 13]

In this framework, pragmatics is seen as the most important function, followed by semantics and lastly by syntax. Functional Grammar provides a structure, the predication, that encompasses the syntactic, semantic, and pragmatic meanings of all of the constituents of a sentence. The primary emphasis in Functional Grammar is on the meaning of each constituent, not on where it is located in the sentence. Only after the constituents' meanings have been established is syntactic placement

given consideration. This is done through the application of expression rules.

A. PREDICATIONS

Predications form the basic components of Functional Grammar. Each predication is based on a single predicate from the lexicon. The lexicon consists of the basic terms of the grammar and the basic predicate frames. The basic terms and predicate frames are described below. A predication describes a complete thought. More often this is a sentence, but it may be a partial sentence, as, for example, in a compound sentence. The predication is expressed by means of a structure called a predicate-frame.

(1) [p A1(X1) A2(X2) ... An(Xn)]
CAT

The predicate-frame provides the following information.

* The predicate. (p)
* The category of the predicate. (Verb, Noun, Adjective)
* The argument positions. (X1, X2, ..., Xn)
* The semantic function of each argument. (Agent, Goal, Recipient, etc.)
* The selection restrictions for each argument. (A1, A2, ..., An)

The predicate-frame in (1) is a nuclear predication. This means it has the minimum number of arguments and their types which are needed to express a complete thought for a given predicate. For example,

(2) give (X1:Animate(X1)) (X2)
VERB AGENT GOAL

(X3:Animate(X3))
REC

The predicate is 'give', of category VERB. There are three arguments; X1, X2, and X3, whose semantic functions are AGENT, GOAL, and RECIPIENT, respectively. Additionally, X1 and X3 are restricted to being animate objects. A predication in the form of (2) is said to be an open predication. It gives all of the semantic arguments and their attributes that are required to form a complete sentence using that verb. When all of the arguments have been filled, the predication is considered fully specified.

A predication may have more semantic functions added to it for further clarification. This is done through satellites. Satellites are specified in the same manner as the arguments in (1) and they would be represented by Y1,Y2,...,Yn, along with B1 ... Bm which represent the selection restrictions for the satellites.

(3) {[$_{\text{CAT}}$p A1(X1) A2(X2) ... An(Xn)] B1(Y1)

B2(Y2) ... Bm(Ym)}

The semantic functions of the arguments together with the verb provide a state of affairs for the sentence.

There are four states of affairs: action, position, process, and state. These are defined by two processes, (±)control and (±)dynamism. These are shown in Table 2.1. Control implies that a being in the sentence controls what is happening. Dynamism implies that something is taking place as opposed to describing a situation. Verbs, such as run and stand, may be used in more than one state of affairs. It is their relationship to the semantic functions of the arguments

that defines the state of affairs. In sentence (4), Bill controls the action in a dynamic setting. In sentence (5), Bob controls the act of standing, but is in a static setting. In sentence (6), the refrigerator does not control, but it is a continuing or dynamic situation. In sentence (7), the car does not control its color and (7) merely describes a static situation. [Ref. 2:pp. 25-39]

(4) Bill ran down the street. (action)

(5) Bob stood on the corner. (position)

(6) The refrigerator is running. (process)

(7) The car is blue. (state).

Table 2.1 States of Affairs

	controlled	uncontrolled
dynamic	action	process
non-dynamic	position	state

B. TERMS

The argument slots are filled with terms, which are found in the lexicon of the grammar. Terms are nouns, verbs, and adjectives. They are defined thus:

(8) (wXi : p(Xi))

w is a term operator which describes whether the term is definite or indefinite, singular or plural. p(Xi) is a predication. The phrase 'the ten butterflies' would be expressed:

(9) $(\iota_{N}dXi : butterfly(Xi))$

Terms may be modified by use of referents. Referents normally take the form of adjectives and are added to the term in the following way.

(10) (wXi : p1(Xi): p2(Xi): ... pn(Xi))

In the phrase 'the ten bright orange butterflies', the referents are 'bright' and 'orange'. The phrase would be expressed as:

(11) (10d Xi: butterfly (Xi): orange (Xi): bright (Xi))
N A A

This phrase could then be inserted into an argument slot of a predication.

C. SEMANTIC FUNCTION HIERARCHY

Dik has established what he terms a Semantic Function Hierarchy (SFH). The hierarchy establishes a relationship between various syntactic and semantic functions that is language independent. The ordering/hierarchy is as follows:

Agent)Goal)Recipient)Beneficiary)Instrument)Location)Temp

Each noun term in a sentence is assigned a semantic function. Additionally, one of the terms is also assigned the syntactic function SUBJECT. Once that term has been identified, its semantic function is marked in the SFH. Then, if a syntactic function OBJECT exists in the sentence, the word which has this function must have a semantic function that is to the right of the semantic function that was marked for SUBJECT. Each language may have sentences that place the SUBJECT in various positions in the hierarchy, but a cut-off point is generally established where asssignment of SUBJECT to semantic functions beyond that point results in poorly formed or nonsensical sentences. The cut-off point for English is BENEFICIARY. At or near the cut-off point,

it is more difficult to find sentences that are "good English". The following sentences illustrate some of the possible SUBJECT and OBJECT assignments. The last sentence illustrates a sentence that tries to go beyond the cut-off point. It is clearly a poorly formed sentence.

a. Bill gave the bread to Tom .
AG-SUBJ GO-OBJ REC

b. Bill gave Tom the bread .
AG-SUBJ REC-OBJ GO

c. Bill bought Tom the bread .
AG-SUBJ BEN-OBJ GO

d. The bread was given to Tom by Bill .
GO-SUBJ REC-OBJ AG

e. Tom was given the bread by Bill .
REC-SUBJ GO AG

f. Tom was bought the bread by Bill .
BEN-SUBJ GO AG

g. In the kitchen was brought the bread for Bill .
LOC-SUBJ GO REC

These relationships are shown in Table 2.2.[Ref. 2:pp. 70-75]

Table 2.2 Subject-Object Relationship

	Semantic Functions			
	AGENT	GOAL	REC	BEN
a	SUBJ	OBJ		
b	SUBJ		OBJ	
c	SUBJ			OBJ
d		SUBJ		
e			SUBJ	
f				SUBJ

D. PRAGMATIC FUNCTIONS

A predication that has been assigned semantic functions, syntactic functions, and a state of affairs appears fully specified. But certain situations are not represented. Consider the following sentences.

(12) BILL drove to Chicago.

(13) Bill DROVE to Chicago.

(14) Bill drove TO Chicago.

(15) Bill drove to CHICAGO.

By emphasizing a different word in each sentence, a different meaning is achieved. To account for such differences, it is necessary to consider the speaker's context, his assessment of what he means, the addressee's assessment of what he has heard, intonation, etc. Functional Grammar provides a set of four pragmatic functions to deal with these situations. The assignment of pragmatic functions to a predication will result in a fully specified predication.

The four pragmatic functions are TOPIC, FOCUS, THEME, and TAIL. The latter two are external to the predication while the first two are internal to the predication. The pragmatic function THEME describes the universe of discourse of a given predication. It is normally associated with left-dislocated phrases such as:

(16) That girl, I like her.

The second external pragmatic function, TAIL, describes an afterthought or something that clarifies the predication. It is normally associated with right-dislocated phrases, such as:

(17) She's a nice lady, my wife.

Functional Grammar assumes that THEME and TAIL are external to the

predication and it uses the following representational schema:

(18) (Xi)THEME, Predication, (Xj)TAIL

where Xi and Xj are FG representations of the phrases.

Internal to the predication, Functional Grammar utilizes two pragmatic functions. TOPIC describes a constituent about which the predication predicates something. FOCUS represents the relatively most important information with respect to the pragmatic concerns of the speaker and the addressee. TOPIC and FOCUS may be assigned to any constituent in the predication, including the verb. A sentence does not necessarily have all four pragmatic functions assigned. Most often, only the internal functions will be assigned.

It is in the treatment of a text or sentence grouping that the external functions play a major role in Functional Grammar. Consider the following paragraph.

(19) John gave Mary a book. Mary gave Bill some money. Bill gave Tom a coat.

Analyzing the first sentence by itself might result in various assignments. John might be assigned as TOPIC and Mary as FOCUS. Book could also be assigned as FOCUS. The assignment might depend on the speaker's intonation. However, looking at all three sentences together reveals several possible combinations of TOPIC and FOCUS. The common thread of this paragraph is the act of giving and thus provides us with the THEME. The THEME could change over time as more sentences are added.

(20) They all wanted to help someone.

(21) They all had finally repaid their debts.

The addition of sentence (20) would change the THEME to 'generosity'. Adding sentence (21) instead, the THEME is probably 'paybacks'. One of the strengths of Functional Grammar lies in the ability to look at a group of sentences and provide an overall meaning to the predications. [Ref. 2:pp. 127-132]

With the addition of the pragmatic functions, it is possible to obtain a fully specified predication. The following example using sentence (22) shows how a sentence is taken from its sentential form to a fully specified predication.

(22) John gave the big red book to the sweet little girl on Tuesday.

The sentence is based on the predication for 'give'.

```
(23) give [(X1:Animate(X1))    (X2)
          V                    AG      GO

           (X3:Animate(X3))    ]
                               REC ACTION
```

Adding the satellite (Y1:time-period(Y1)$_{TIME}$ to (23) results in the extended predication:

```
(24) {give [(X1:Animate(X1))   (X2)
           V                   AG     GO

        (X3:Animate(X3))    ]       (Y1:time-per(Y1))     }
                            REC ACTION                TIME
```

The terms are then inserted into (24) resulting in the following predication.

```
(25) {give [(d1X1:John (X1))
           V            N      AG-SUBJ

      (d1X2:book (X2)(d1X4:red (X3)(d1X5:big (X5))))
                 N              A              A        GO

      (d1X3:girl (X3)(d1X6:little (X6)
                 N                 A

      (d1X7:sweet (X7))))    ]       (d1Y1:Tuesday (Y1))     }
                  A        REC ACTION              N      TIME
```

Next the pragmatic functions, TOPIC and FOCUS, are assigned to (25) resulting in the fully specified predication:

```
(26) {give [(d1X1:John (X1))
        V          N     AG-SUBJ-TOP

     (d1X2:book (X2)(d1X4:red (X3)(d1X5:big (X5)))
               N            A            A      GO

     (d1X3:girl (X3)(d1X6:little (X6)
               N               A

     (d1X7:sweet (X7))))       ]      (d1Y1:Tuesday (Y1))    }
              A       REC-FOC ACTION            N     TIME
```

E. EXPRESSION RULES

Once a fully specified predication has been achieved, a means for mapping the elements of the predication onto a linguistic expression is required. A set of language dependent expression rules provide a means of accomplishing this. Although many types of expression rules exist, they can be generally divided into three groups: case marking, word order, and intonation. These rules work together to form the linguistic expression.

1. Case Marking

Each of the syntactic, semantic, pragmatic, and operator markings gives a clue as to how the sentence will appear. The term operator (1dX1: elephant) would map to 'the'. Had it been (2dX1: elephant), it would have mapped into 'the two' and 'elephant' would change to 'elephants'. Using syntactic and semantic marking and the Semantic Function Hierarchy provides more rules. For example, if a term is marked as AGENT but not SUBJECT, the preposition 'by' will be mapped onto the term. Having a term marked as AGENT and SUBJECT would indicate

that the sentence begins with the SUBJ-AGENT and therefore does not need the preposition 'by'. Verbs are also affected by case marking. The different tenses and the use of auxiliary verbs are triggered by expression rules.[Ref. 2:pp. 158-161]

2. Word Order

Functional Grammar provides a language independent preferred order of constituents (LIPOC) which form another section of the expression rules. LIPOC can be expressed as

PROcl < PRO < NP < NPP < V < NP < PNP < SUB

where:

PROcl = Clitic Pronoun
PRO = Pronoun
NP = Noun Phrase
NPP = Postpositional Noun Phrase
V = Verb
PNP = Prepositional Noun Phrase
SUB = Subordinate Clause

An example of where this ordering is apparent is shown below.

(27) The man in the uniform gave a ticket to the boy.

where:

NP = The man
NPP = in the uniform
V = gave
NP = a ticket
PNP = to the boy

Most sentences do not contain all of the constituents, but the constituents that are present conform to LIPOC. Additionally, languages have a syntactic ordering. The ordering of English is Subject-Verb-Object (SVO).[Ref. 2:pp. 192-194]

3. Intonation

The pragmatic function TOPIC and FOCUS provide the information to give the proper intonation to the sentence. The predication in (1) would result in the sentence in (2).

(28) {drive [(d1X1:John)]
V-FOC N AG-SUBJ-TOP ACTION

(Y1:Chicago) }
N LOC

(29) John DROVE to Chicago.

Of course, sentence (29) is not the only interpretation of the predication in (28). In (29), the past tense of give was used. (30) and (31) are also possible interpretations.

(30) John WILL DRIVE to Chicago.

(31) John DRIVES to Chicago.

These rules can be grouped together to provide a means of mapping the predication onto the linguistic expression. Although not in standard Backus Naur Form (BNF), the rules follow a similar format. The following rules illustrate this concept.

-iX1 -) the indifinite article "a" or "an".

-dX1 -) the definite article "the".

-d2X1 -) the number "two".

Figure 2.1 summarizes the organization of a Functional Grammar and shows how the concepts outlined above are related. [Ref. 2:p. 23]

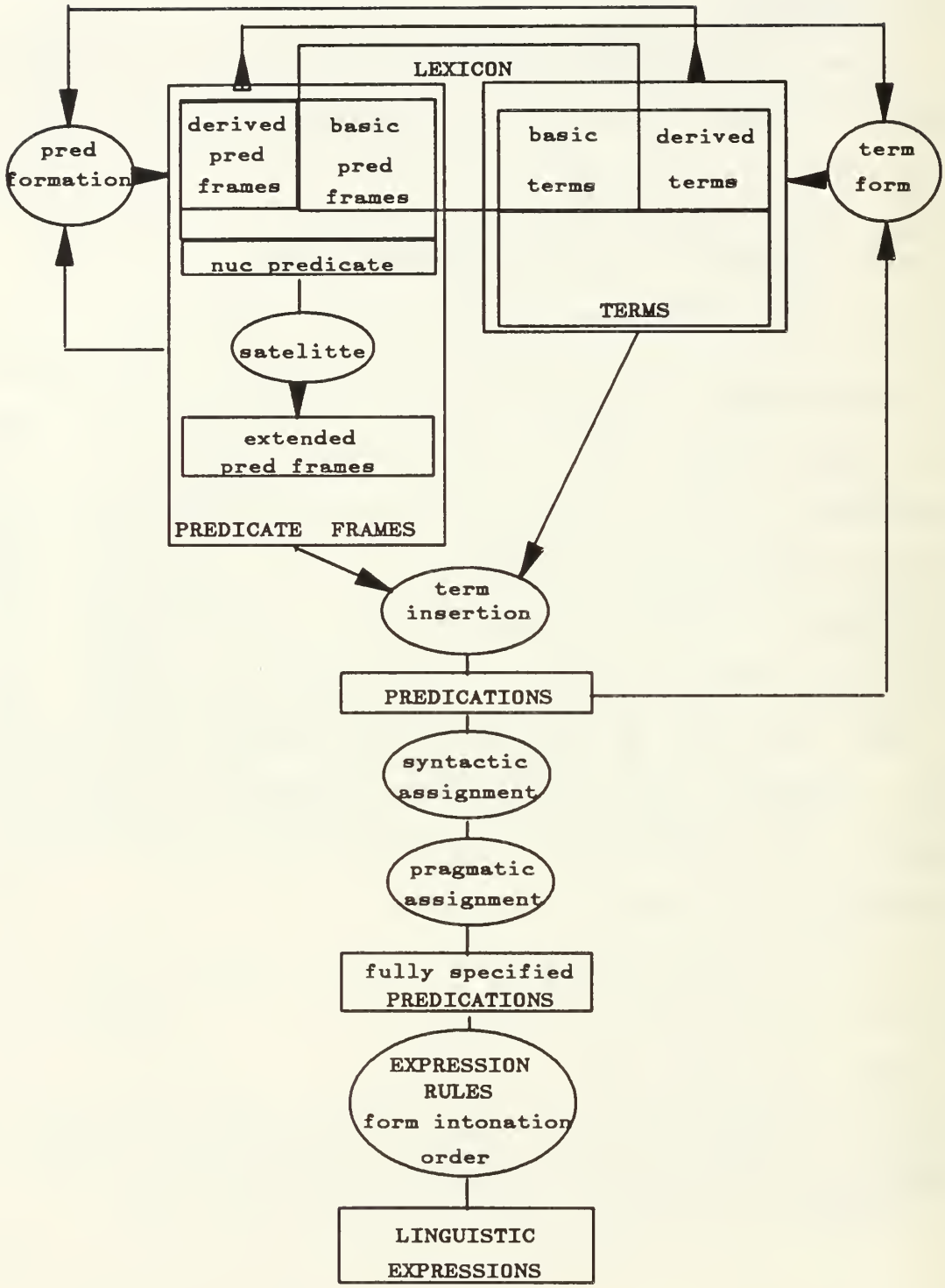

FIGURE 2.2 FUNCTIONAL GRAMMAR ORGANIZATION

III. PROGRAM DESCRIPTION

To develop a program which will process natural language using Functional Grammar requires looking at Functional Grammar from a completely different view than was explained in Chapter II. In Chapter II, the placement of terms into their proper syntactic, semantic, and pragmatic argument positions was done from a linguist's perspective. A linguist selects the various functions by looking at the sentence and deciding intuitively what role each word in the sentence plays. He accomplishes this by means of his background in language. It is the duplication of this thinking process in the computer that the program which has been developed attempts to attain. Many constraints must be imposed in order to keep this project manageable. The approach taken and constraints imposed are discussed below. The program follows the general flow shown in Figure 3.1. It is designed to read a paragraph, convert each sentence into Functional Grammar notation, and then ascertain the pragmatic constituents of the sentence, in particular the overall theme of the paragraph.

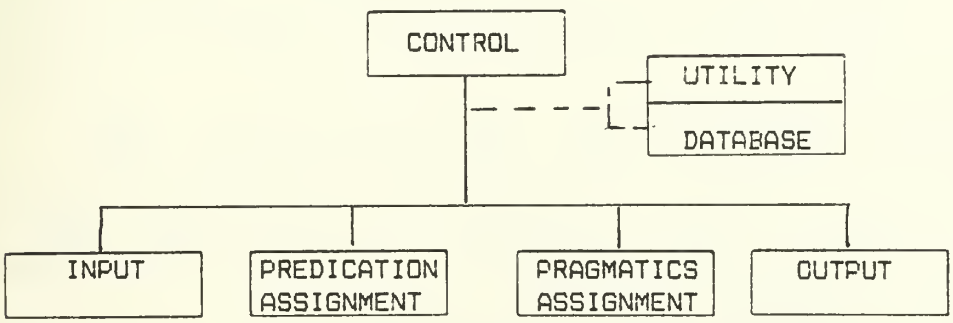

Figure 3.1 Program Flow

A. APPROACH

1. Control

The CONTROL module acts as the traffic director of the program. It starts the program, prompts the user to get the input file, and sends the input file to the INPUT module. The INPUT module is a straightforward application of an input routine found in Programming in Prolog [Ref. 8], which places the paragraph into a list of lists. Each sentence is a list and each word or punctuation mark is an element of that list. When the list is complete, the CONTROL module sends the list to the PREDICATION module where each sentence is converted into Functional Grammar notation. The predication is then passed to the PRAGMATICS module where the pragmatic functions are assigned. Control then sends the resulting predication to the OUTPUT module for presentation to the user.

2. Predication Assignment

The PREDICATION module receives the list of sentences and puts them into Functional Grammar notation. Doing this requires that the sentence be looked at word by word. In a Functional Grammar, each sentence is based on a verb, which determines the required semantic arguments. Terms are then inserted into the semantic argument slots. This provides a convenient division of the sentence for processing. The sentence can be looked at as a series of clauses, each clause containing a noun term. In sentence (1), the clauses have been underlined to illustrate the division.

(1) John gave the book to Mary in the library.

The first task of the program is to divide the sentence into clauses and then process each clause in turn. A term list is maintained throughout the processing of the sentence to store the clauses. A Predication list is also maintained throughout the processing. The term list has the following format.

[Term, Code, Adj]

Code is further formatted as follows.

[Syn, Sem, Def, Num, Prep, Pos]

where:

Syn = syntax (noun, verb, adjective)

Sem = semantics (agent, goal, recipient, ..etc)

Def = definite/indefinite

Num = number

Prep = preposition

Pos = position

The 'Adj' slot is used to store adjectives or referent clauses. A noun term is placed into the list with the term in the 'Term' slot, an 'n' in the 'Syn' slot, a number in the 'Position' slot, and an empty set indicator in the 'Adj' slot. The rest of the slots contain 'z' for 'not assigned'. After a clause is processed, many of the slots will be filled.

Once the term has been placed in the term list, the clause leading up to the term is processed word by word. If the word is an adjective, it is placed in the 'Adj' slot. The adjective has the same format as the noun term, that is, [adjective, Code, Adj]. Determiners, numbers, and prepositions are all stored in the code list of the

current term. For example, the partial sentence (2) results in the term notation in (3).

(2) The red box in the drawer...

(3) [box,[n,z,d,z,z,3],[red,[a,z,z,z,z,2],

[drawer,[n,z,d,z,in,6],[]]]]

As can be seen, adjectives and referent phrases can nested in the 'Adj' slot. Adjectives are placed in the term list in the clause placement module. Referent phrases are placed there later.

If a word is a verb, the open predication for that verb is placed in the predication list. The open predication is in the following format.

[verb,[v,state,z,z,z,pos],[att,sem,att,sem,...att,sem]]

The code list is similar to the one for noun terms. The third element of the verb list is Semantics. It contains each of the verb's semantic arguments with their corresponding attributes. For example, the semantics for the verb 'give' is shown below.

* [human,ag,any,go,animate,rec]

Particular circumstances will be encountered that require special processing. These can be discovered at the clause-process level. If a sentence contains a series of terms, such as in sentence (4), the series will be treated as a single term.

(4) John, Jack, and Bill have new cars.

Since a series of words used in this manner must perform the same semantic function, the program processes the clause using the first term of the series and then recombines the terms after the clause is processed.

In some instances, the pragmatic functions THEME and TAIL are readily apparent and should be processed as such. Left dislocated sentences, such as (5) show the theme. Right disclocated sentences, such as (6) give the Tail.

(5) As for Bill, he already owns a car.

(6) I like her, my wife.

When these situations are encountered the relevant clause will be saved outside of the term list.

When all of the terms have been placed in the term list and the verbs have been placed in the predication list, two resolution checks are made. The predication list is checked for multiple predicates and the term list is checked for pronouns. If a sentence has more than one verb as in sentence (7), one of the verbs must be an auxilary verb. The auxiliary verb is therefore not needed and is discarded and the remaining verb is used for the predication.

(7) John was given the book by Mary.

In the case of sentence (7), give is used. Consequently the auxiliary verb (was) is discarded.

Next, the term list is searched for pronouns. Pronouns are used frequently in natural language. If used properly, the term they are used in place of will often be apparent. In the case of nominative pronouns (referred to as case 1 pronouns), the most likely referent is the AGENT of the previous sentence. This is illustrated in sentences (8) and (9).

(8) John carried the book home.

(9) He read it after dinner.

In the case of objective pronouns (case 2 pronouns), a similar relation exists, this time with the recipient or beneficiary of the previous statement as shown in sentences (10) and (11).

(10) John gave the book to Bill.

(11) It was an ideal present for him.

The use of 'it' in (9) and (11) should be noted. In most situations, 'it' will refer to the goal of the previous sentence. In both pairs of sentences 'it' referred to 'book'. The program replaces the pronouns with the terms they represent in accordance with the guidelines mentioned above.

The program is now ready for syntactic and semantic function assignment, which are done concurrently. It is accomplished by the set of 'assign' rules. The rules were designed with the following principles in mind.

* SVO- English is an SVO language (Subject-Verb-Object). Therefore all sentences will have their syntactic functions in that order.
* SFH - As discussed in Chapter II, there are definite positioning rules and relationships between the syntactic and semantic functions.
* Attributes- every semantic function has certain attributes to which it must adhere. For example, in the predication for 'give', the AGENT must be a human. This is an implementation dependent restriction. These attributes can be clearly defined.
* Prepositions- Each semantic function uses a distinct set of prepositions. Although the sets are not mutually exclusive, they lower the range of possibilities. The prepositions used for this

program's database and their semantic uses are shown in Figure 3.2.

* Referents- A clause positioned after the subject clause but before the verb necessarily refers to the subject.

Prepositions

	A	A	A	A	A	B	B	B	B	B	B	D	F	F	I	I	I	N	O	O	P	T	T	T	U	U	W	W	W
	B	B	F	G	T	E	E	E	E	E	Y	U	O	R	N	N	N	E	N	V	A	H	O	O	N	N	I	I	I
	O	O	T	A		F	H	L	N	S		R	R	O		S	T	A		E	S	R		W	D	T	T	T	T
	U	V	E	I		O	I	O	E	I		I		M		I	O	R		R	T	O		A	E	I	H	H	H
	T	E	R	S		R	N	W	A	D		N				D						U		R	R	L		I	O
				T		E	D		T	E		G				E						G		D				N	U
									H													H							T

AG									X																				
GOAL																													
REC															X				X										
BEN											X						X												
INST																									X		X		
SOR												X																	
DIR																						X	X						
TIME	X		X		X	X				X	X	X	X	X				X	X	X	X			X					
LOC		X		X	X		X	X	X	X	X				X	X		X	X	X	X				X		X		
Ø																													

Figure 3.2 Preposition-Semantic Relationships

Using all of the above principles, the semantic and syntactic functions can be pinpointed. An example of the use of the principles follows using sentence (12).

(12) The man in the library gave the book to Mary.

Prior to entering the function assignment module, the program will have provided the following terms.

[man, [n, z, d, z, z, 2], []]

[library, [n, z, d, z, in, 5], []]

[book, [n, z, d, z, z, , 8], []]

[Mary, [n, z, d, z, z, to, 10], []]

The predicate will be:

[give,[v,action,z,z,z,6],[human,AG,any,GO,animate,REC]]

The terms are looked at in the order they appear in the sentence.

The first term (man), has a position before the verb and its attribute matches all semantic functions. Its position makes it the SUBJECT, but at this point its semantic function is unresolved. It is assigned temporarily as SUBJECT and as AGENT, since AGENT is the first semantic function. The second term (library), also has a position before the verb. Since the first term is the SUBJECT, the second term acts as a referent to the first term. It is therefore placed in the 'Adj' slot of the term 'man'. This results in the following partiallly filled predication.

[give,[v,action,z,z,z,6],

[[man,[n,z,d,z,z,2],

[[library,[n,z,d,z,in,5],[]]],SUBJAG,

any,GO,animate,REC]

The third clause (book), is the first clause after the verb, which is the object position. An OBJECT is assigned when the semantic function AGENT acts as SUBJECT, in accordance with Table 2.2. Since that is the case in this example, 'book' is assigned as OBJECT. Additionally, 'book' matches only the attribute of the semantic function GOAL. Any RECIPIENT or AGENT used after the verb must always have a preposition. 'Book' does not have a preposition. It is therefore assigned the semantic function of GOAL. The last term (Mary), has a position after the verb, matches all atributes, and has the preposition 'to'. Figure 3.2 reveals that 'to' is used with RECIPIENT, DIRECTION, and TIME. Mary

therefore must be assigned as RECIPIENT, since the other semantic functions do not exist in this predication. The resultant predication is shown below.

```
[give,[v,action,z,z,z,6],
    [[man,[n,z,d,z,z,2],
          [[library,[n,z,d,z,in,5],[]]],SUBJAG,
     [book,[n,z,d,z,z,8],[]],OBJGO,
     [Mary,[n,z,d,z,z,to,10],[]],REC]]
```

It should be noted that the program makes assignments temporarily as was the case in the assignment above of the first term. In this regard, Prolog's backtracking techniques provide the proper mechanism to achieve the trial and error method required. At any point in the assignment process that a match cannot be made, the program backtracks until it finds a good permutation. Should there be more terms left over when the predicate's semantic functions have been assigned, these terms will be assigned as satellites and placed in the extended predication. Once the predication has been assigned, it is sent to the PRAGMATICS ASSIGNMENT module.

Syntactic/Semantic Functions

		SUBJ	OBJ	AG	GO	REC	BEN
	TOPIC	X		X			
	TOPIC			X			
	FOCUS	X			X		
Pragmatic	FOCUS	X				X	
Functions	FOCUS	X					X
	FOCUS		X		X		
	FOCUS		X			X	
	FOCUS		X				X

Figure 3.3 Internal Pragmatic Assignments

3. Pragmatics Assignment

The internal pragmatic functions, TOPIC and FOCUS, are directly related to the syntactic and semantic assignments. If a term is assigned as AGENT as well as SUBJECT or as just the AGENT with no syntactic function, that term is most likely the TOPIC. Figure 3.3 shows other relationships which indicate a probable pragmatic function assignment.

The external pragmatic functions, THEME and TAIL, are assigned quite differently. Certainly, if a THEME clause or a TAIL clause was passed into this section of the program, as discussed earlier, that clause would automatically be assigned the THEME or TAIL accordingly. In most cases, however, this will not be the case. Without the benefit of intonation in written work, a word count approach is used. At the end of each sentence, the nouns, verbs, and adjectives are counted. They are placed in a list which is maintained throughout the program. As new words are used, they are added to the list. As words are used subsequently, their count is increased. Additionally, concept words are examined. Many words suggest a concept. For instance, the words happy, grin, playground, and giggle all suggest the concept of pleasure. After each word is added to the list or updated, it is compared to the concept lists. If it appears there, that concept is added to the word count list. At the end of each sentence, the word count list is sorted and the first two words on the list become the THEME and TAIL, respectively. This may seem rather arbitrary and it is, but it must be understood that external pragmatic functions are just that, external. An individual sentence may not provide an external function. The

external functions are normally found by examining a group of several sentences to determine an overall subject. It is this subject for which the program is searching.

B. CONSTRAINTS

The design of a natural language processor using Functional Grammar is an ambitious one. To maintain a scope that is large enough to demonstrate the principles of Functional Grammar yet small enough to complete in a limited timeframe required the imposition of several constraints. This program will process simple and compound sentences, but only ones with simple grammatical expressions. Sentences using verbs in the infinitive form or sentences using more than two verbs such as sentence (13) are not supported.

(13) The book is to be given to Mary by John.

Prepositions were limited to the ones shown in Figure 3.2. Compound prepositions such as 'according to' or 'because of' and phrasal prepositions such as 'as far as' or 'in spite of' were not used.

As was explained in Chapter II, Functional Grammar utilizes four states of affairs. This program processes only those verbs in the action and state groups. Action was chosen because it is the state that is discussed most often in Functional Grammar literature and it accounts for most of the sentences in the English language. State was a logical second choice because it represents those verbs that are non-dynamic and uncontrolled, which is the opposite of action verbs. Reducing the states of affairs also reduced the semantic functions. Dik discusses some 24 different functions, but this program is limited to

the 10 functions shown in Figure 3.2. Many words can have more than one syntactic meaning. A common example is the group of words ending in 'ing', which may be used as adjectives, verbs, or nouns. This program only addresses single usage of these words. To limit the number of verb forms, only past tense verbs were used. The database for the program was made only large enough to support the above constraints.

These constraints do not represent a failure of FG to process the whole of natural language. The extensive backtracking and look ahead facilities to process the many compound terms, odd constructs, etc. are available in Prolog. It is necessary only to write the many rules required to cover all situations. However, the trade-off is program size. The program would be a very large one, with an equally large database, a situation typical of almost all natural language processors.

IV. PROGRAM RESULTS

To demonstrate the capability of this program the text used must test the various sentence configurations and provide an overall theme for a given paragraph. Three test runs are discussed which accomplish this task. The computer output for these tests are found in Appendices B, C, and D. The following paragraph was used for the first test.

> The setting was a brisk, Autumn day. The park was near the river. Jack carried his gun in his pocket. As for his partner, Bill watched the playground from the opposite side. Fallen leaves were on the ground. The man in the playground raised his hand. Bill placed his gloved hand into his pocket. Jack strained his eyes. He noticed the ominous briefcase on the ground. He heard the ticking. He waited for the explosion. The waiting was unnerving.

Table 4.1 Pragmatic Assignments

	Pragmatic Functions			
Sentence	Focus	Topic	Theme	Tail
1	setting	none	day	setting
2	park	none	suspense	river
3	gun	Jack	pocket	gun
4	playground	Bill	partner	pleasure
5	leaves	none	pleasure	ground
6	hand	man	pleasure	playground
7	hand	Bill	pleasure	suspense
8	eyes	Jack	suspense	pleasure
9	briefcase	Jack	suspense	pleasure
10	ticking	Jack	suspense	Jack
11	explosion	Jack	suspense	Jack
12	waiting	none	suspense	Jack

Sentences of different types are used and the entire paragraph sets a mood of suspense without ever actually stating it. The program assigns the pragmatic functions at the end of each sentence. The

results of the first test are found in Appendix B. A listing of the pragmatic assignments is provided in Table 4.1.

The internal function assignments are, for the most part, obvious assignments. In some cases no assignment was made. This occurs in sentences whose semantic and syntactic assignments do not readily point to a particular pragmatic function. Sentences such as this are commonly found in the 'state' state of affairs. Consider the last statement of the paragraph.

* The waiting was unnerving.

The sentence is about waiting. 'Unnerving' modifies 'waiting'. 'Waiting' is the most important information in the sentence, thus fulfulling the requirement for FOCUS. If unnerving modifies waiting and waiting is the FOCUS, which term is the TOPIC? Is there a term which presents the entity about which the predicate predicates something per Dik's definition of TOPIC? Waiting appears to be a candidate for both functions. Although there is no set rule which disallows a term from holding both internal pragmatic functions, it seems unnecessary, at best. Functional Grammar literature includes few examples of pragmatic functions in other than the action state of affairs and thus leaves open the method of assignment in the other states of affairs. It seems plausible that there are situations, such as the one described above, where the pragmatic function FOCUS is sufficient to describe the situation.

There are other cases in 'state' predications that are unclear. Consider the second sentence.

* The park was near the river.

The program assigned 'park' as FOCUS, using the same reasoning as above. The clause 'near the river' certainly modifies 'the park'. With no further information, the assignment is probably valid. However, add an accent to 'near the river' and the meaning changes. Is the FOCUS now 'near the river'? Is the TOPIC now 'the park'? Both possible assignments have merit. Without any extra information, the assignment of FOCUS is sufficient to show the meaning of the predication.

External functions were arrived at by a word count. As can be seen in Table 4.1, the THEME and TAIL changed as the program progressed through the paragraph, finally settling on 'suspense' and 'Jack', respectively, near the end of the program. This was the correct interpretation. A reader should come away from the paragraph with the overall theme of a suspense situation with Jack as the most prominent character.

The second test run used the same test paragraph as the first test run. However, the sixth, ninth, and the tenth sentences were changed to the passive voice. As can be seen by comparing the output of this test, found in Appendix C, to the results of the first test, the only changes were in the syntactic assignments in the changed sentences. The pragmatic assignments remained the same as those in the first run.

The third test run, Appendix D, also used the text from the first run, but changed the order of sentences. Although the THEME and TAIL assignments were different from those in the first run due to the rearrangement, they were identical from the fifth sentence through the final assignment. The rearranged paragraph is as follows.

The man in playground raised his hand. Jack carried his gun in his pocket. As for his partner, Bill watched the playground from the opposite side. The setting was a brisk, Autumn day. The park was near the river. Fallen leaves were on the ground. Bill placed his gloved hand into his pocket. Jack strained his eyes. He noticed the ominous briefcase on the ground. He heard the ticking. He waited for the explosion. The waiting was unnerving.

Although the program arrived at the proper external pragmatic assignments, the assignments early in the paragraph are inaccurate. What is not shown in Table 4.1 is that until the ninth sentence no THEME had a clear cut majority. In some cases, THEME and TAIL were equal, with the determining factor being which word came first in the word count list. A weighted count was considered, but rejected for lack of a valid weight system. Terms that are assigned syntactic functions as well as semantic functions should get greater consideration for pragmatic assignment. This, however, does not adequately address verbs. Verbs form a very important part of our language and need to be considered. Looking at prose in general, it would appear that concept words would very often be the type of word that would be found in the final THEME assignment. Therefore any weighting system must consider concept words. Many such issues must be addressed before an adequate weighting system can be attained. In general, each word and group of sentences must be looked at individually with no weighting system. In this specific case, the method this program uses for assigning internal pragmatic functions is adequate, providing a large number of sentences are being evaluated.

Expanding this program by adding an interactive question and answer section would greatly enhance its capability. This would give the program intonation of words which provide more clues to the assignment

of the pragmatic functions. Additionally, this program is essentially a pattern-matching mechanism designed to take advantage of some of the more obvious grammar rules of the English language. In addition to these rules, a set of inference rules could be added whereby the combination of syntactic, semantic and internal pragmatic functions over a 2-3 sentence range would suggest a possible theme. It must be noted that these improvements bring with them a large overhead. Asking questions means having to accept a larger number of words as input, thereby increasing the size of the database. More importantly, the design of the questions would involve an extremely large number of individual cases. Knowing which question to ask is as important as getting the right answer.

V. CONCLUSIONS

What should be the purpose of producing a natural language processor? Research in this area has produced processors that evaluate many sentences, each sentence evoking a particlular response from the computer. Even some micro-computer game programs provide limited 'natural language' responses to computer generated questions. Given Dik's premise that a grammar should be a means of social interaction, a processor should be able to read a generous amount of text and return some insightful meaning that the user might otherwise not have noticed.

This paper presented a program which shows the feasibility of using Functional Grammar in natural language processing. Reading a paragraph and producing a THEME is a small achievement when compared to the types of applications that are possible. One possible application of this procedure is as a psychologist's assistant. Consider the task of dream anyalysis. Dreams may suggest to a psychologist certain reasons for a patient's problems. However, the theme of the patient's dreams may be abstract or obscure and present an enigma to the psychologist. This program would require a large database combined with a program capable of interaction between the user and database. The interaction is necessary to provide the intonation missing in a straight text analyzer. Knowing which word is emphasized in a sentence will help determine the meaning of the sentence. Having such a tool may evoke concepts that were not obvious to the psychologist. These concepts

might hold the key to the patient's disorder or at least provide a new avenue to explore.

Functional Grammar is a suitable vehicle for pursuing such projects. It is Functional Grammar's view of language as a means of social interaction that makes it such an attractive method for natural language processing.

```
/******************** CONTROL MODULE ************************/

/* The CONTROL module first consults all applicable files.          */
/* It then calls the INPUT module which reads in the file,          */
/* the PREDICATION module which forms the predication,and           */
/* the OUTPUT file which prints out the results.                    */
/*******************************************************************/


go:- fileread,introduction(Text,Out),see(Text),sentreview(Sentlist),seen,
   predication(Sentlist,[],P,[],Wordcount,[],Matrix),
   output(Out,Matrix,P,Wordcount).
fileread:- consult(input),consult(predication),consult(utility),
   consult(database),consult(pragmatics),consult(output).
sentreview(S):- get0(C),sent(C,S).
sent(C,[]):- lastsent(C).
sent(C,[S|S1]):- read_in(C,S),sentreview(S1).

predication([],P,P,Wc,Wc,Mat,Mat).
predication([F|R],P,P4,Wc,Wc2,Oldmat,Newmat):-
   clause_process(F,F,P,[],P1,T1,[],Prag,1),resolve(P1,T1,P2,T2),
   funct_assign(P2,T2,P3),pragmatics(P3,Wc,Wc1,Prag,Oldmat,Intmat),
   predication(R,P3,P4,Wc1,Wc2,Intmat,Newmat).

introduction(Text,Out):- space(10),query(Text,Out),space(10).

query(Text,Out):- printstring("This is a natural language processor"),
   printstring(" which uses Functional Grammar."),nl,nl,
   printstring("To use, enter the name of the file you wish to have evaluated"),
   nl,printstring("followed by a period."),nl,read(Text),nl,nl,
   printstring("Enter the name of the output file followed by a period."),
   nl,read(Out).
```

```
/************************** INPUT ********************************/

/* The INPUT module reads from a text file. It reads in a sentence      */
/* at a time. Each sentence is placed in a list, with each word or      */
/* punctuation mark being one element of the list.                      */
/************************************************************************/


read_in(C,[W|Ws]):- readword(C,W,C1),restsent(W,C1,Ws).

restsent(W,_,[]):- lastword(W),!.
restsent(W,C,[W1|Ws]):- readword(C,W1,C1), restsent(W1,C1,Ws).

readword(C,W,C1):- single_character(C),!,name(W,[C]), get0(C1).
readword(C,W,C2):- in_word(C,NewC),!,get0(C1),restword(C1,Cs,C2),
                   name(W,[NewC|Cs]).
readword(C,W,C2):- get0(C1), readword(C1,W,C2).

restword(C,[NewC|Cs],C2):- in_word(C,NewC),!,get0(C1),restword(C1,Cs,C2).
restword(C,[],C).

single_character(44).  /* , */
single_character(59).  /* ; */
single_character(58).  /* : */
single_character(63).  /* ? */
single_character(33).  /* ! */
single_character(46).  /* . */

in_word(C,C):- C>96, C<123.                 /*a b ...z */
in_word(C,L):- C>64, C<91, L is C+32.       /*A B ...Z */
in_word(C,C):- C>47, C<58.                  /*1 2 ...9 */
in_word(39,39).                             /* ' */
in_word(45,45).                             /* - */

lastword('.').
lastword('!').
lastword('?').

lastsent(26).
```

```
/****************** PREDICATION DEVELOPMENT ******************/

/* The PREDICATION DEVELOPMENT Module takes as input sentences  */
/* from the INPUT module. The sentence is first checked to see if it is a  */
/* question or a declaration. If it is a question, it is put into declaration  */
/* form. The sentence is then passed to the CLAUSE PROCESS section  */
/* where the program looks at each word until it finds a term. When a  */
/* term is found, the clause upto and including the term is processed,  */
/* which defines the term clause. After all of the term clauses have been  */
/* defined, the predication is built in the FUNCTION ASSIGNMENT  */
/* module, where the syntactic and semantic assignments are made. The  */
/* predication is then sent to the PRAGMATICS ASSIGNMENT module,  */
/* where the pragmatic functions are assigned and the complete predication  */
/* is defined. It is then sent to the OUTPUT module where it is printed  */
/* on the screen.  */
/*****************************************************************/


check_quest([W,Is,N|R],[N,Is|R]):- question([W,Is,N|R]),what(W).
check_quest([Is,N|R],[N,Is|R]):- question([Is,N|R]).

question([?|R]).
question([.|R]):- fail,!.
question([!|R]):- fail,!.
question([F|R]):- question(R).
```

```
/************************ CLAUSE PROCESS ************************/
/* The CLAUSE PROCESS module takes a sentence and puts it into  */
/* Functional Grammar notation. The output will consist of two lists, the */
/* TERMLIST and the PREDICATION. The TERMLIST is a list of the  */
/* terms in the following format.                               */
/*   [Term,[Code],[Ref],Term,[Code],[Ref],...,Term,[Code],[Ref]] */
/* The Term is the term as found in the sentence. The Code is a list of */
/* parameters in the following format.                          */
/*   [Syntax-Semantics-Number-(Definite/indefinite)-Preposition-Position] */
/* Ref is a list of referents (adjectives) which are listed in the same */
/* format as the termlist.                                      */
/* The sentence is searched until a term is found. Then the clause up to */
/* and including the term is looked at word by word in the word-process */
/* submodule. If a word is an adjective, it is placed after the term in */
/* the Ref section. If it is a preposition, it is placed in the */
/* preposition section of the Code. If it is a number ,article, or a */
/* determinator, the appropriate sections of the Code are changed. The */
/* Code is initiated with "z" in each section. This will signify that the */
/* term does not have an item to fill a particular section. If a verb is */
/* found, the open predication for that verb is placed in the PREDICATION */
/* list. In the case of apparent Theme or Tail, the module will store the */
/* applicable term in the variable 'Prag'. This clause will then not be */
/* processed with the rest of the sentence.                     */
/****************************************************************/

clause_process(Osent,[First|Sent],Pred1,Term1,Pred1,Term1,Prag,Prag,P):-
   end_of_sentence(First).
clause_process(Osent,[First|Sent],P1,[W,C,A|T1],P2,T4,Prag,Prag2,Pos):-
   conj(First),no_more_verbs(Sent),
   build_series(Osent,[First|Sent],P1,[W,C,A],T2,Pos,Pos1),Pos2 is Pos1-1,
   align_preps(T2,PrepT3),red_clause(Osent,Pos2,Sent1),
   clause_process(Osent,Sent1,P1,[T3|T1],P2,T4,Prag,Prag2,Pos1).

clause_process(Osent,[First|Sent],P1,[W,C,A|T1],P2,T4,Prag,Prag2,Pos):-
   conj(First),no_prev_verbs(0,Pos,Osent),
   build_series(Osent,[First|Sent],P1,[W,C,A],T2,Pos,Pos1),Pos2 is Pos1-1,
   align_preps(T2,Prep,T3),red_clause(Osent,Pos2,Sent1),
   clause_process(Osent,Sent1,P1,[T3|T1],P2,T4,Prag,Prag2,Pos1).

clause_process(Osent,[','|R],P1,[W,C,A|T1],P2,T4,Prag,Prag2,Pos):- series_search(R)
   build_series(Osent,[','|R],P1,[W,C,A],T2,Pos,Pos1),
   align_preps(T2,Prep,T3),Pos2 is Pos1-1,red_clause(Osent,Pos2,R1),
   clause_process(Osent,R1,P1,[T3|T1],P2,T4,Prag,Prag2,Pos1).
clause_process(Osent,[','|R],P1,T1,P2,T2,Prag,Prag2,Pos):- length(T1,L),
```

```
    (L=3),make_theme(T1,Prag1),Pos1 is Pos+1,
    clause_process(Osent,R,P1,[],P2,T2,Prag1,Prag2,Pos1).
clause_process(Osent,[',',|R],P1,T1,P2,T2,Prag,Prag1,Pos):- make_tail(R,Prag1,Pos).
clause_process(Osent,Sent,Pred1,Term1,Predication,Termlist,Prag,Prag2,P):-
    findterm(Sent,P,P1),select(Osent,P1,Term),
    make_term(Term1,P1,Term,Termlist1),
    word_proc(Sent,Pred1,Termlist1,V1,T1,P),
    red_clause(Osent,P1,Sent1),P2 is P1+1,
    clause_process(Osent,Sent1,V1,T1,Predication,Termlist,Prag,Prag2,P2).

series_search([',',or|R]).
series_search([',',and|R]).
series_search([F,S|R]):- series_search([S|R]).

build_series(Osent,[Conj|R],P1,T1,T3,Pos,Pos3):- conj(Conj),Pos1 is Pos+1,
    findterm(R,Pos1,Pos2),select(Osent,Pos2,Term),make_term(T1,Pos2,Term,T2),
    word_proc(R,P1,T2,P2,T3,Pos1),
    Pos3 is Pos2+1.
build_series(Osent,[',',Conj|R],P1,T1,T3,Pos,Pos3):- conj(Conj),Pos1 is Pos+2,
    findterm(R,Pos1,Pos2),select(Osent,Pos2,Term),make_term(T1,Pos2,Term,T2),
    word_proc(R,P1,T2,P2,T3,Pos1),
    Pos3 is Pos2+1.
build_series(Osent,[',',|R],P1,T1,T4,Pos,Pos4):- Pos1 is Pos+1,
    findterm(R,Pos1,Pos2),select(Osent,Pos2,Term),make_term(T1,Pos2,Term,T2),
    word_proc(R,P1,T2,P2,T3,Pos1),red_clause(Osent,Pos2,Sent1),
    Pos3 is Pos2+1,build_series(Osent,Sent1,P2,T3,T4,Pos3,Pos4).

make_theme([Term,[Syn,Sem|R]|R1],[Term,[Syn,theme|R]|R1]).
make_tail(R,Tail,Pos):- clause_process(R,[F|R1],[],[],P2,Tail,[],Pos).

align_preps([Trm,[Sy,Sm,D,N,Prp|R],Ad|R1],Prp,[Trm,[Sy,Sm,D,N,Prp|R],Ad|R1]):-
    null(R1).
align_preps([Trm,[Sy,Sm,D,N,A|R],Ad|R1],Prp,[Trm,[Sy,Sm,D,N,Prp|R],Ad|R1]):-
    align_preps(R1,Prp,R1).

findterm([First|Rest],Pos,Pos):- is_term(First).
findterm([F|S],P,P1):- P2 is P+1,findterm(S,P2,P1).

make_term(Termlist,P,First,[First,[n,z,z,z,z,P],[]|Termlist]).

word_proc([F|S],V,T,V,T,P):- last_term(F,T).
word_proc([F|S],V,[T,Cod|R],V1,T1,P):- is_determiner(F,D),
    change_list(D,4,Cod,Cod1),P1 is P+1,word_proc(S,V,[T,Cod1|R],V1,T1,P1).
word_proc([F|S],V,[T,Cod|R],V1,T1,P):- is_number(F,N),change_list(N,3,Cod,Cod1)
```

```
    change_list(d,4,Cod1,Cod2),P1 is P+1,
    word_proc(S,V,[T,Cod2|R],V1,T1,P1).
word_proc([F|S],V,[T,Cod,Adj|R],V1,T1,P):- is_adverb(F),P1 is P+1,
    word_proc(S,V,[T,Cod,Adj,F,[ad,z,z,z,ad,P],[]|R],V1,T1,P1).
word_proc([F|S],V,[Ta,Tb,Adj|R],V1,T1,P):- is_adjective(F),
    make_adj(F,P,Adj,Adj1),add_adj(S,S1,P,P1,Adj1,Adj2),Pos is P1+1,
    word_proc(S1,V,[Ta,Tb,Adj2|R],V1,T1,Pos).
word_proc([F|S],V,[T,Cod|R],V1,T1,P):- is_prep(F),change_list(F,5,Cod,Cod1),
    P1 is P+1,word_proc(S,V,[T,Cod1|R],V1,T1,P1).
word_proc([F|S],V,T,V1,T2,P):- is_verb(F,Verb),make_pred(Verb,P,Pred,T,T1),
    P1 is P+1,word_proc(S,[Pred|V],T1,V1,T2,P1).

fill_term(A,W):- attribute(A,L),member(W,L).
fill_term(A,W):- a_kind_of(A,B),fill_term(B,W).

last_term(First,[First|Rest]).

make_pred(Verb,1,Pred,T1,T2):- make_term(T1,1,you,T2),find_pred(Verb,2,Pred).
make_pred(Verb,P,Pred,T1,T1):- find_pred(Verb,P,Pred).

add_adj([',',A,B|R],Sent,P,P1,Adj1,Adj2):- Pos is P+2,make_adj(A,Pos,Adj1,Adj2),
    add_adj([B|R],Sent,Pos,P1,Adj2,Adj3).
add_adj(R,R,P,P,A,A).

make_adj(Term,P,[],[Term,[a,z,z,z,z,P],[]]).
make_adj(Term,P,[A1,C,Adj],[A1,C,Newadj]):- make_adj(Term,P,Adj,Newadj).

find_pred(Verb,P,[Verb,[v,State,0,z,z,P],Semantics]):-
    pred(Verb,State,Semantics).

no_more_verbs([]).
no_more_verbs([F|R]):- is_verb(F,Any),!,fail.
no_more_verbs([F|R]):- no_more_verbs(R).

no_prev_verbs(N,N,Sent).
no_prev_verbs(N,N1,[F|R]):- is_verb(F,Any),!,fail.
no_prev_verbs(N,N1,[F|R]):- N2 is N+1,no_prev_verbs(N2,N1,R).

end_of_sentence(.).
```

```
/****************** PREDICATION RESOLUTION ****************/

/* The PREDICATION RESOLUTION module takes as input the        */
/* PREDICATION list and the TERMLIST. It searches the TERMLIST */
/* for pronouns. If a pronoun is found, it is changed to the proper term  */
/* that it refers to, according to the pronoun rules. Then the            */
/* PREDICATION list is scanned to see if there is more than one verb.  */
/* If so, then one of the verbs must be an auxiliary verb and is deleted.  */
/* The output is the new TERMLIST and the new PREDICATION.     */
/*************************************************************/

resolve(P,T,P1,T1):- mult_pred(P,P1),pronouns(P1,T,T1).

mult_pred([[V,[A,B,0|R]|R1]|R2],[[V,[A,B,1|R]|R1]|R2]):- aux(V),last_pred(R2).
mult_pred([[V,[A,B,0|R]|R1],[V1,[D,E,0|R3]|R4]|R2],[[V,[A,B,1|R]|R1]|R2]):-
   aux(V1).
mult_pred([[V,[A,B,0|R]|R1]|R2],[[V,[A,B,1|R]|R1]|R2]).

last_pred([]).
last_pred([[V,[A,B,1|R]|R1]|R2]).
more_pred([[V,[A,B,0|R]|R1]|R2]).
add_one([[V,[A,B,0|R]|R1]|R2],[[V,[A,B,1|R]|R1]|R2]).

pronouns(V,[],[]).

pronouns(V,[[T1,C,Adj|R]|R1],[[Term,C,Adj|R2]|R3]):- case1(T1),
   find_last_sem(V,ag,Term),pronouns(V,R,R2),pronouns(V,R1,R3).
pronouns(V,[[T1,C,Adj|R]|R1],[[Term,C,Adj|R2]|R3]):- case2(T1),
   find_last_sem(V,rec,Term),pronouns(V,R,R2),pronouns(V,R1,R3).
pronouns(V,[[T1,C,Adj|R]|R1],[[Term,C,Adj|R2]|R3]):- case2(T1),
   find_last_sem(V,ben,Term),pronouns(V,R,R2),pronouns(V,R1,R3).

pronouns(V,[it,C,Adj|R],[Term,C,Adj|R1]):- find_last_sem(V,go,Term),
   pronouns(V,R,R1).
pronouns(V,[T1,C,Adj|R],[Term,C,Adj|R1]):- case1(T1),
   find_last_sem(V,subjag,Term),pronouns(V,R,R1).
pronouns(V,[T1,C,Adj|R],[Term,C,Adj|R1]):- case1(T1),find_last_sem(V,ag,Term),
   pronouns(V,R,R1).
pronouns(V,[T1,C,Adj|R],[Term,C,Adj|R1]):- case2(T1),find_last_sem(V,rec,Term),
   pronouns(V,R,R1).
pronouns(V,[T1,C,Adj|R],[Term,C,Adj|R1]):- case2(T1),
   find_last_sem(V,objrec,Term),pronouns(V,R,R1).
pronouns(V,[T1,C,Adj|R],[Term,C,Adj|R1]):- case2(T1),
   find_last_sem(V,subjrec,Term),pronouns(V,R,R1).
```

```
pronouns(V,[T1,C,Adj|R],[Term,C,Adj|R1]):- case2(T1),find_last_sem(V,ben,Term),
   pronouns(V,R,R1).
pronouns(V,[T1,C,Adj|R],[Term,C,Adj|R1]):- case2(T1),
   find_last_sem(V,objben,Term),pronouns(V,R,R1).
pronouns(V,[T1,C,Adj|R],[Term,C,Adj|R1]):- case2(T1),
   find_last_sem(V,subjben,Term),pronouns(V,R,R1).

pronouns(V,[[T,C,Adj|R]|R1],[[T,C,Adj|R2]|R3]):- pronouns(V,R,R2),
   pronouns(V,R1,R3).
pronouns(V,[T,C,Adj|R],[T,C,Adj|R1]):- pronouns(V,R,R1).


find_last_sem([Verb1|Rest],Type,Term):- last_sent(Rest,Type,Term).
last_sent([[Verb,Code,Sem|R]|R1],Type,Term):- search_sem(Sem,Type,Term).
search_sem([[T,C,A],S|R],S,T).
search_sem([A,S|R],Type,Term):- search_sem(R,Type,Term).




/******************** FUNCTION  ASSIGNMENT ********************/

/* The FUNCTION ASSIGNMENT module takes as input the TERMLIST */
/* and the PREDICATION. Using the preposition rules, the attribute      */
/* matching rules, and the semantic function hierarchy rules, the semantics  */
/* and syntax are assigned. The TERMLIST is looked at term by term.     */
/* Each term is compared to each term in the PREDICATION until a        */
/* match is found. The term is first compared to the attribute of each term  */
/* open predication. When a match is found, then the preposition is     */
/* checked for appropriateness. If it is ok, then the syntax rules are  */
/* applied, and the resultant semantic and syntactic assignment is      */
/* applied. If at any point in the above procedure there is no match,   */
/* that term is abandoned and the next one is checked. Any extra        */
/* clauses after the predication is filled are defined as satellites    */
/* and are placed in the extended predication.                          */
/*********************************************************************/

funct_assign([[Verb,Code,Semantics]|R],T,[[Verb,Code,Newsem,Ext]|R]):-
   select(Code,6,Verbpos),rev_term(T,[],T1),
   assign(T1,Semantics,Verbpos,1,Newsem,[],Ext).


assign([],S,V,C,S,E,E).
```

```
assign([[Term,Code,Adj|R]|R1],Semantics,Verbpos,Ctr,Newsem,Ext1,Ext3):-
    assign([Term,Code,Adj|R1],Semantics,Verbpos,Ctr,Sem1,Ext1,Ext2),
    add_back_series([Term,Code,Adj|R],Sem1,Newsem,Ext2,Ext3).

assign([Term,Cod,Adj|R],Semantics,Verbpos,Ctr,Newsem,Ext1,Ext3):-
    select(Cod,5,ad),extend([Term,Cod,Adj],Ext1,Ext2),Ctr1 is Ctr+1,
    assign(R,Semantics,Verbpos,Ctr1,Newsem,Ext2,Ext3).

assign([Term,Code,Adj|R],Semantics,Verbpos,1,Newsem,Ext1,Ext2):-
    select(Code,6,Pos),Pos<Verbpos,select(Code,5,Prep),
    attri_match(Term,Code,Adj,Semantics,Prep,Sem1),Ctr1 is 1+1,
    assign(R,Sem1,Verbpos,Ctr1,Newsem,Ext1,Ext2).

assign([Term,Code,Adj|R],Semantics,Verbpos,Ctr,Newsem,Ext1,Ext2):-
    select(Code,6,Pos),Pos<Verbpos,referent([Term,Code,Adj],Semantics,Sem1),
    Ctr1 is Ctr+1,assign(R,Sem1,Verbpos,Ctr1,Newsem,Ext1,Ext2).

assign([Term,Code,Adj|R],Semantics,Verbpos,Ctr,Newsem,Ext1,Ext3):-
    is_the_subjag(Semantics),is_obj_assgn(Semantics),is_sem_filled(Semantics),
    extend([Term,Code,Adj],Ext1,Ext2),Ctr1 is Ctr+1,
    assign(R,Semantics,Verbpos,Ctr1,Newsem,Ext2,Ext3).

assign([Term,Code,Adj|R],Semantics,Verbpos,Ctr,Newsem,Ext1,Ext2):-
    is_the_subjag(Semantics),is_obj_assgn(Semantics),select(Code,5,Prep),
    att_place([Term,Code,Adj],Prep,Semantics,Sem1),Ctr1 is Ctr+1,
    assign(R,Sem1,Verbpos,Ctr,Newsem,Ext1,Ext2).

assign([Term,Code,Adj|R],Semantics,Verbpos,Ctr,Newsem,Ext1,Ext2):-
    is_the_subjag(Semantics),
    select(Code,5,Prep),obj_assign([Term,Code,Adj],Prep,Semantics,Sem1),
    Ctr1 is Ctr+1,assign(R,Sem1,Verbpos,Ctr1,Newsem,Ext1,Ext2).

assign([Term,Code,Adj|R],Semantics,Verbpos,Ctr,Newsem,Ext1,Ext3):-
    is_sem_filled(Semantics),
    extend([Term,Code,Adj],Ext1,Ext2),Ctr1 is Ctr+1,
    assign(R,Semantics,Verbpos,Ctr1,Newsem,Ext2,Ext3).

assign([Term,Code,Adj|R],Semantics,Verbpos,Ctr,Newsem,Ext1,Ext2):-
    select(Code,5,Prep),
    att_place([Term,Code,Adj],Prep,Semantics,Sem1),Ctr1 is Ctr+1,
    assign(R,Sem1,Verbpos,Ctr,Newsem,Ext1,Ext2).

assign([Term,Code,Adj|R],Semantics,Verbpos,Ctr,Newsem,Ext1,Ext3):-
    extend([Term,Code,Adj],Ext1,Ext2),Ctr1 is Ctr+1,
```

```
    assign(R,Semantics,Verbpos,Ctr1,Newsem,Ext2,Ext3).

attri_match(Term,Code,Adj,[Att,Sem|R],Prep,[[Term,Code,Adj],Newsem|R]):-
    is_attribute(Term,Att),prep(Prep,Sem),subj(Sem,Newsem).
attri_match(Term,Code,Adj,[Att,Sem|R],Prep,[Att,Sem|R1]):-
    attri_match(Term,Code,Adj,R,Prep,R1).

att_place([Term,Code,Adj],Prep,[Att,Sem|R],[[Term,Code,Adj],Newsem|R]):-
    is_attribute(Term,Att),prep(Prep,Newsem).
att_place(Term,Prep,[Att,Sem|R],[Att,Sem|R1]):-att_place(Term,Prep,R,R1).

subj(ag,subjag).
subj(rec,subjrec).
subj(ben,subjben).
subj(go,subjgo).
subj(0,subj0).
obj(go,objgo).
obj(rec,objrec).
obj(ben,objben).

referent(Term,[Word,Sem|R],[Word1,Sem|R]):- subj(Any,Sem),
    add_ref(Word,Term,Word1).
referent(Term,[Word,Sem|R],[Word,Sem|R1]):- referent(Term,R,R1).

add_ref([Term,Code,[]],Ref,[Term,Code,Ref]).
add_ref([Term,Code,Adj],Ref,[Term,Code,Adj1]):- add_ref(Adj,Ref,Adj1).

is_the_subjag([Att,subjag|R]).
is_the_subjag([Att,Sem|R]):- is_the_subjag(R).

is_obj_assgn([Att,Sem|R]):- obj(Any,Sem).
is_obj_assgn([Att,Sem|R]):- is_obj_assgn(R).

is_sem_filled([]).
is_sem_filled([Att,Sem|R]):- is_list(Att),is_sem_filled(R).

extend([Term,Code,Adj],[],[[Term,Code,Adj],adverb]):- select(Code,5,ad).
extend([Term,Code,Adj],[],[[Term,Code,Adj],Sem]):- select(Code,5,Prep),
    prep(Prep,Sem),sem_attribute(Sem,Att),is_attribute(Term,Att).
extend(Term,[Sat1,Sem|R],[Sat1,Sem|R1]):- extend(Term,R,R1).

obj_assign([Term,Code,Adj],Prep,[Att,Sem|R],[[Term,Code,Adj],Newsem|R]):-
    is_attribute(Term,Att),prep(Prep,Sem),obj(Sem,Newsem).
obj_assign(Term,Prep,[Att,Sem|R],[Att,Sem|R1]):- obj_assign(Term,Prep,R,R1).
```

```
renew(P,P).

add_back_series([Term,Code,Adj|R],[[Term,Code,Adj],Sem|R1],[[[Term,Code,Adj|R]|
    Sem|R1],Ext1,Ext1).
add_back_series([Term,Code,Adj|R],[],[],[[Term,Code,Adj],Sem|R1],
    [[[Term,Code,Adj|R]],Sem|R1]).
add_back_series([Term,Code,Adj|R],[],R2,[[A,B,C],Sem|R1],[[A,B,C],Sem|R3]):-
    add_back_series([Term,Code,Adj|R],[],R2,R1,R3).
add_back_series([Term,Code,Adj|R],[[A,B,C],Sem|R1],[[A,B,C],Sem|R2],Ext1,Ext2):-
    add_back_series([Term,Code,Adj|R],R1,R2,Ext1,Ext2).
```

```
/****************** PRAGMATICS ASSIGNMENT ******************/

/* The PRAGMATICS ASSIGNMENT module recieves the predication   */
/* for the sentence plus the running wordcount. It first does an update   */
/* of the wordcount. Next it assigns the pragmatic functions. This        */
/* is done by first determining the internal functions and then the       */
/* external functions. Topic and Focus are determined by looking          */
/* at which words have both a syntactic as well as semantic               */
/* function. Theme and Tail are determined by searching the word          */
/* count list to see which concepts are most utilized. When the           */
/* pragmatics have been assigned, they are stored in a running            */
/* matrix which maintains by sentence the following information.          */
/*                  Sent #/Focus/Topic/Theme/Tail                         */
/************************************************************************/


pragmatics(Predication,Wc,Wc2,Prags,Oldmatrix,Newmatrix):-
   word_count(Predication,Prags,Wc,Wc1),
   prag_assign(Predication,Wc1,Wc2,Prags,Focus,Topic,Theme,Tail),
   save_prags(Oldmatrix,Focus,Topic,Theme,Tail,Newmatrix).


/********************** WORD COUNT ************************/

word_count([[Verb,Code,Semantics|R]|R1],Prags,Wc,Wc6):-
   check_and_add(Verb,Wc,Wc1),
   concept_add(Verb,Wc1,Wc2),term_count(Semantics,Wc2,Wc3),
   term_count(R,Wc3,Wc4),adj_count(Prags,Wc4,Wc5),
   remove(Wc5,Wc6).

term_count([[]],Wc,Wc).
term_count([],Wc,Wc).
term_count([[[Term,Cod,Adj],Sem|R]],Wc,Wc3):- check_and_add(Term,Wc,Wc1),
   concept_add(Term,Wc1,Wc2),term_count(R,Wc2,Wc3).
term_count([[[Term,Cod,Adj|R]],Sem|R1],Wc,Wc5):-check_and_add(Term,Wc,Wc1),
   concept_add(Term,Wc1,Wc2),adj_count(Adj,Wc2,Wc3),
   term_count(R,Wc3,Wc4),term_count(R1,Wc4,Wc5).
term_count([[Term,Code,Adj],Sem|R],Wc,Wc4):- check_and_add(Term,Wc,Wc1),
   concept_add(Term,Wc1,Wc2),adj_count(Adj,Wc2,Wc3),
   term_count(R,Wc3,Wc4).

check_and_add([],Wc,Wc).
check_and_add(Word,[Word,Count|R],[Word,Ct1|R]):- Ct1 is Count + 1.
check_and_add(Word,[],[Word,1]).
```

```
check_and_add(Word,[Other,Count|R],[Other,Count|R1]):-
   check_and_add(Word,R,R1).

adj_count([],Wc,Wc).
adj_count([Term,Cod,Adj],Wc,Wc3):- check_and_add(Term,Wc,Wc1),
   concept_add(Term,Wc1,Wc2),adj_count(Adj,Wc2,Wc3).

concept_add(Term,Wc,Wc1):- is_like(Term,Concept),
   check_and_add(Concept,Wc,Wc1).
concept_add(Term,Wc,Wc).

remove([],[]).
remove([Word,Count|R],R1):- case3(Word),remove(R,R1).
remove([Word,Count|R],R1):- aux(Word),remove(R,R1).
remove([Word,Count|R],[Word,Count|R1]):- remove(R,R1).


/*************** PRAGMATICS ASSIGNMENT *****************/

prag_assign([[Verb,Code,Semantics|R]|R1],Wc,Wc1,Prags,Focus,Topic,Theme,Tail):-
   internal(Semantics,Focus,Topic),word_sort(Wc,[],Wc1),
   external(Wc1,Prags,Theme,Tail).

internal(Sem,Focus,Topic):- foc_find(Sem,Focus),top_find(Sem,Topic).

foc_find([],none).
foc_find([[Term,Code,Adj],Sem|R],Term):- foc_match(Sem).
foc_find([Term,Sem|R],Focus):- foc_find(R,Focus).

foc_match(objgo).
foc_match(objrec).
foc_match(objben).
foc_match(subjgo).
foc_match(subjrec).
foc_match(subjben).
foc_match(subj0).

top_find([],none).
top_find([[Term,Code,Adj],Sem|R],Term):- top_match(Sem).
top_find([Term,Sem|R],Topic):- top_find(R,Topic).

top_match(subjag).
top_match(ag).
```

```
external([Theme,N1,Tail,N2|R],[],Theme,Tail).
external([Tail,N1|R],[Theme,[S,theme|R1]|R2],Theme,Tail):- not(Tail=Theme).
external([W1,N1,Tail,N2|R],[Theme,[S,theme|R1]|R2],Theme,Tail).
external([Theme,N1|R],[Tail|R1],Theme,Tail):- not(Theme=Tail).
external([W1,N1,Theme,N2|R],[Tail|R1],Theme,Tail).


/*************** SAVE PRAGMATICS IN MATRIX ***************/

save_prags([],Focus,Topic,Theme,Tail,[[1,Focus,Topic,Theme,Tail]]).
save_prags(Matrix,Focus,Topic,Theme,Tail,Final):- first_of_first(Matrix,Num),
   Num1 is Num+1,append([[Num1,Focus,Topic,Theme,Tail]],Matrix,Final).

first_of_first([First|Rest],Number):- select(First,1,Number).
```

```
/************************* OUTPUT *************************/

/* The OUTPUT module takes the Pragmatics matrix, the Word     */
/* count list, and the Predication from the PRAGMATICS         */
/* ASSIGNMENT module The information is formatted so that the  */
/* pragmatic functions are printed for each sentence, the      */
/* predication verb and semantic functions are printed, and    */
/* the final word count is printed.                            */
/***************************************************************/


output(Out,Matrix,Pred,Wc):- tell(Out),nl,nl,nl,title,heading,matrix(Matrix),
    nl,nl,pred_label,nl,pred_print(Pred),nl,nl,told.

title:- tab(15),first(Title),printstring(Title),nl,tab(15),
    underline(Line),bord(Line,8),nl,nl.

heading:- second(Heading),printstring(Heading),secadd(Head1),
    printstring(Head1),nl,third(More),printstring(More),
    thiradd(Mor1),printstring(Mor1),nl,nl,nl,label(Label),rhh(Label),nl.

matrix([]).
matrix([First|Rest]):- matrix(Rest),rhh(First).

first("Pragmatic Assessment of Paragraph").
underline('----').
second("The paragraph submitted has been transormed").
secadd("into Functional Grammar notation.").
third("The pragmatic functions were then determined").
thiradd("and are provided below").
label(['Sentence','Focus','Topic','Theme','Tail']).

rhh([]):-nl.
rhh([H|T]):- col_print(H),rhh(T).

phh([]):- nl.
phh([H|T]):- write(H),tab(1),phh(T).

bord(Word,0):-nl.
bord(Word,Count):- write(Word),Ct1 is Count-1,bord(Word,Ct1).

space(0):- nl.
space(S1):- nl,S2 is S1-1,space(S2).
```

```
pred_print([]).
pred_print([[Verb,Code,Semantics,Ext]|R]):- nl,underline(Line),
    write(Line),nl,write(Verb),
    nuc_print(Semantics),ext_print(Ext),pred_print(R).


nuc_print([]):- nl.
nuc_print([[[Term,Code,Adj|R]],Sem|R1]):- nl,tab(12),col_print(Sem),
    col_print(Term),adj_print(Adj),more_nuc(R),nuc_print(R1).
nuc_print([[Term,Code,Adj],Sem|R]):- nl,tab(12),col_print(Sem),
    col_print(Term),adj_print(Adj),nuc_print(R).

adj_print([]).
adj_print([Term,Code,Adj]):-col_print(Term),adj_print(Adj).

ext_print([]).
ext_print([[Term,Code,Adj],Sem|R]):- tab(12),col_print(Sem),
    col_print(Term),adj_print(Adj),ext_print(R).


pad(0).
pad(N):- tab(1),N1 is N-1,pad(N1).

col_print(Word):- write(Word),name(Word,List),length(List,Len),N is 12-Len,
    pad(N).

printstring([]).
printstring([H|T]):- put(H),printstring(T).

more_nuc([]).
more_nuc([Term,Code,Adj|R]):- nl,tab(12),col_print(Term),
    adj_print(Adj),more_nuc(R).

pred_label:- lab(Label),rhh(Label),nl.

lab(['VERB','SEM/SYN','TERM','REFERENTS']).
```

```
/************************ UTILITY ****************************/

/* The UTILITY module contains several list processing rules        */
/* which are used throughout the other modules.                     */
/********************************************************************/

member(X,[]):- fail,!.
member(X,[X|L]).
member(X,[Y|L]):- not(X=Y),member(X,L).

sem_member(F,[]):- fail.
sem_member(F,[[[F|R]]|R1]).
sem_member(F,[[F|R]|R1]).
sem_member(F,[[T1|R],Sem|R1]):- not(F=T1),sem_member(F,R1).

insert_item([],[],L,L).
insert_item(X,Y,[X|L],[Y|L]).
insert_item(X,Y,[Z|L],[Z|L1]):- not(X=Z),insert_item(X,Y,L,L1).

change_list(X,1,[Y|L],[X|L]).
change_list(X,Z,[Y|L],[Y|L1]):- Z1 is Z-1,change_list(X,Z1,L,L1).

append([],L,L).
append([X|L1],L2,[X|L3]):- append(L1,L2,L3).

select([X|L],1,X).
select([X|L],I,Y):- I1 is I-1,select(L,I1,Y).

red_clause(Sent,0,Sent).
red_clause([F|Sent],P,Sent1):- P1 is P-1,red_clause(Sent,P1,Sent1).

is_list([]).
is_list([_|_]).

rev_term([],M,M).
rev_term([T,C,A|R],M,L):- rev_term(R,[T,C,A|M],L).

word_sort([],N,N).
word_sort([W,C|R],Old,New):- sort(W,C,Old,Int),word_sort(R,Int,New).

sort(W1,C1,[],[W1,C1]).
sort(W1,C1,[W2,C2|R1],[W1,C1,W2,C2|R1]):- C1>=C2.
sort(W1,C1,[W2,C2|R1],[W2,C2|R2]):- sort(W1,C1,R1,R2).
```

```
/************************* DATABASE *************************/

/* The DATABASE module contains the words and their relationships */
/* necessary to adequately process a section of text. This database */
/* is limited to the words found in the test text. It is also limited */
/* to past tense and to words in the action and process states of */
/* affairs. A limited number of semantic cases are used. These cases */
/* are the most common cases found in the states of affairs used. */
/************************************************************/

a_kind_of(animate,human).
a_kind_of(animate,animal).
a_kind_of(sor,loc).

a_type_of(human,[mary,john,jack,bill,man,woman,child,boy,girl,partner]).
a_type_of(animal,[dog,cat,horse,cow]).
a_type_of(tim,[hour,day,minute,year,while,hours,days,minutes,years]).
a_type_of(dir,[north,south,east,west,way,park]).
a_type_of(loc,[library,street,house,building,park,river,bridge,playground,
    side,ground,pocket]).
a_type_of(thing,[book,leaves,hand,eyes,briefcase,gun]).
a_type_of(event,[setting,ticking,explosion,waiting,unnerving]).

is_term(F):- a_type_of(A,L),member(F,L).
is_term(F):- pro_list(L),member(F,L).
is_adjective(F):- adj_list(L),member(F,L).
is_adjective(F):- possessive(L),member(F,L).
is_prep(F):- prep_list(L),member(F,L).
is_pron(F):- pro_list(L),member(F,L).
is_attribute(adverb,adverb).
is_attribute(A,any).
is_attribute(A,B):- a_type_of(B,L),member(A,L).
is_attribute(A,B):- a_kind_of(B,C),is_attribute(A,C).
is_adverb(F):- adverb_list(L),member(F,L).

is_determiner(the,d).
is_determiner(this,d).
is_determiner(that,d).
is_determiner(a,d).
is_determiner(an,d).
is_determiner(some,i).
is_determiner(these,d).
is_determiner(those,d).
```

```
is_number(one,1).
is_number(two,2).
is_number(three,3).
is_number(four,4).
is_number(five,5).
is_number(six,6).
is_number(seven,7).
is_number(eight,8).
is_number(nine,9).
is_number(ten,10).

conj(and).
conj(or).
conj(but).

adj_list([brisk,autumn,opposite,fallen,gloved,ominous,mine,yours,his,hers,ours,
    theirs]).

prep_list([by,to,from,in,near,across,towards,as,for,on,into]).

pro_list([i,you,he,she,we,they,it,me,him,her,us,them,myself,yourself,
    himself,herself,ourselves,themselves,itself]).

adverb_list([quietly,softly,unnerving]).

pred(give,action,[human,ag,any,go,animate,rec]).
pred(drive,action,[human,ag]).
pred(walk,action,[animate,ag]).
pred(approach,action,[animate,ag,loc,loc]).
pred(raise,action,[animate,ag,any,go]).
pred(place,action,[animate,ag,any,go,loc,loc]).
pred(strain,action,[animate,ag,any,go]).
pred(see,action,[animate,ag,any,go]).
pred(notice,action,[animate,ag,any,go]).
pred(hear,action,[animate,ag,any,go]).
pred(wait,action,[animate,ag,any,go]).
pred(is,state,[any,0]).
pred(watch,action,[animate,ag,any,go]).
pred(carry,action,[human,ag,any,go]).

aux(is).
aux(have).
```

```
is_verb(gave,give).
is_verb(was,is).
is_verb(were,is).
is_verb(drove,drive).
is_verb(walked,walk).
is_verb(approached,approach).
is_verb(raised,raise).
is_verb(placed,place).
is_verb(strained,strain).
is_verb(saw,see).
is_verb(heard,hear).
is_verb(waited,wait).
is_verb(watched,watch).
is_verb(carried,carry).
is_verb(noticed,notice).

is_like(Term,Concept):- subject(Concept,List),member(Term,List).
subject(generosity,[give,deliver,loan,lend]).
subject(suspense,[river,briefcase,ticking,explosion,wait,brisk,gloved,
    ominous,raise,strain,wait,quietly,softly,unnerving]).
subject(movement,[drive,walk,approach,raise,place]).
subject(pleasure,[park,river,leaves,playground]).
subject(calmness,[river,leaves,fallen,quietly,softly]).

what(what).
what(who).
what(why).
what(where).
what(when).
what(how).

case1(i).
case1(you).
case1(he).
case1(she).
case1(we).
case1(they).

case2(me).
case2(you).
case2(him).
case2(her).
case2(us).
case2(them).
```

```
case3(mine).
case3(yours).
case3(his).
case3(hers).
case3(ours).
case3(theirs).

prep(about,tim).
prep(after,tim).
prep(before,tim).
prep(during,tim).
prep(through,tim).
prep(until,tim).
prep(above,loc).
prep(against,loc).
prep(behind,loc).
prep(below,loc).
prep(beneath,loc).
prep(beside,loc).
prep(in,loc).
prep(inside,loc).
prep(near,loc).
prep(under,loc).
prep(under,loc).
prep(within,loc).
prep(toward,dir).
prep(with,ins).
prep(without,ins).
prep(into,loc).
prep(at,tim).
prep(at,loc).
prep(in,tim).
prep(in,loc).
prep(over,tim).
prep(over,loc).
prep(past,tim).
prep(past,loc).
prep(for,go).
prep(for,ben).
prep(for,ben).
prep(for,tim).
prep(from,sor).
prep(from,tim).
prep(by,ag).
```

```
prep(by,tim).
prep(by,loc).
prep(by,pro).
prep(to,rec).
prep(to,dir).
prep(to,tim).
prep(on,ben).
prep(on,loc).
prep(ad,adverb).
prep(z,z).
prep(z,A).

sem_attribute(adverb,adverb).
sem_attribute(z,any).
sem_attribute(tim,tim).
sem_attribute(dir,dir).
sem_attribute(loc,loc).
sem_attribute(sor,sor).
add_object(rec,objrec).
add_object(ben,objben).
add_object(go,objgo).
add_object([],[]).
```

APPENDIX B

Pragmatic Assessment of Paragraph

The paragraph submitted has been transormed into Functional Grammar notation. The pragmatic functions were then determined and are provided below.

Sentence	Focus	Topic	Theme	Tail
1	setting	none	day	setting
2	park	none	suspense	river
3	gun	jack	pocket	gun
4	playground	bill	partner	pleasure
5	leaves	none	pleasure	ground
6	hand	man	pleasure	playground
7	hand	bill	pleasure	suspense
8	eyes	jack	suspense	pleasure
9	briefcase	jack	suspense	pleasure
10	ticking	jack	suspense	jack
11	explosion	jack	suspense	jack
12	waiting	none	suspense	jack

PREDICATIONS

VERB	SYN/SEM	TERM	REFERENTS	
is				
	subj0	waiting		
	z	unnerving		
wait				
	subjag	jack		
	objgo	explosion		
hear				
	subjag	jack		
	objgo	ticking		
notice				
	subjag	jack		
	objgo	briefcase	ominous	
	loc	ground		
strain				
	subjag	jack		
	objgo	eyes	his	
place				
	subjag	bill		
	objgo	hand	his	gloved
	loc	pocket	his	

VERB	SYN/SEM	TERM	REFERENTS	
raise				
	subjag	man	playground	
	objgo	hand	his	
is				
	subj0	leaves	fallen	
	loc	ground		
watch				
	subjag	bill		
	objgo	playground		
	sor	side	opposite	
carry				
	subjag	jack		
	objgo	gun	his	
	loc	pocket	his	
is				
	subj0	park		
	loc	river		
is				
	subj0	setting		
	z	day	brisk	autumn

APPENDIX C

Pragmatic Assessment of Paragraph

The paragraph submitted has been transormed into Functional Grammar notation. The pragmatic functions were then determined and are provided below.

Sentence	Focus	Topic	Theme	Tail
1	setting	none	day	setting
2	park	none	suspense	river
3	gun	jack	pocket	gun
4	playground	bill	partner	pleasure
5	leaves	none	pleasure	ground
6	hand	man	pleasure	playground
7	hand	bill	pleasure	suspense
8	eyes	jack	suspense	pleasure
9	briefcase	jack	suspense	pleasure
10	ticking	jack	suspense	jack
11	explosion	jack	suspense	jack
12	waiting	none	suspense	jack

PREDICATIONS

VERB	SEM/SYN	TERM	REFERENTS	
is				
	subj0	waiting		
	z	unnerving		
wait				
	subjag	jack		
	objgo	explosion		
hear				
	ag	jack		
	subjgo	ticking		
notice				
	ag	jack		
	subjgo	briefcase	ominous	
	loc	ground		
strain				
	subjag	jack		
	objgo	eyes	his	
place				
	subjag	bill		
	objgo	hand	his	gloved
	loc	pocket	his	

VERB	SYN/SEM	TERM	REFERENTS	
raise				
	ag	man		
	subjgo	hand		
	loc	playground		
is				
	subj0	leaves	fallen	
	loc	ground		
watch				
	subjag	bill		
	objgo	playground		
	sor	side	opposite	
carry				
	subjag	jack		
	objgo	gun	his	
	loc	pocket	his	
is				
	subj0	park		
	loc	river		
is				
	subj0	setting		
	z	day	brisk	autumn

APPENDIX D

Pragmatic Assessment of Paragraph

The paragraph submitted has been transormed into Functional Grammar notation. The pragmatic functions were then determined and are provided below.

Sentence	Focus	Topic	Theme	Tail
1	hand	man	hand	pleasure
2	gun	jack	pocket	gun
3	playground	bill	partner	pleasure
4	setting	none	playground	pleasure
5	park	none	pleasure	suspense
6	leaves	none	pleasure	playground
7	hand	bill	pleasure	suspense
8	eyes	jack	suspense	pleasure
9	briefcase	jack	suspense	pleasure
10	ticking	jack	suspense	jack
11	explosion	jack	suspense	jack
12	waiting	none	suspense	jack

PREDICATIONS

VERB	SEM/SYN	TERM	REFERENTS	
is				
	subj0	waiting		
	z	unnerving		
wait				
	subjag	jack		
	objgo	explosion		
hear				
	subjag	jack		
	objgo	ticking		
notice				
	subjag	jack		
	objgo	briefcase	ominous	
	loc	ground		
strain				
	subjag	jack		
	objgo	eyes	his	
place				
	subjag	bill		
	objgo	hand	his	gloved
	loc	pocket	his	

VERB	SYN/SEM	TERM	REFERENTS	
is				
	subj0	leaves	fallen	
	loc	ground		
is				
	subj0	park		
	loc	river		
is				
	subj0	setting		
	z	day	brisk	autumn
watch				
	subjag	bill		
	objgo	playground		
	sor	side	opposite	
carry				
	subjag	jack		
	objgo	gun	his	
	loc	pocket	his	
raise				
	subjag	man	playground	
	objgo	hand	his	

LIST OF REFERENCES

1. Winograd, Terry, Understanding Natural Language, Academic Press, 1972.

2. Dik, Simon C., Functional Grammar, Academic Press, 1981.

3. Chomsky, Noam, Syntactic Structures, Mouton & Co., 1957.

4. Chomsky, Noam, Aspects of the Theory of Syntax, The MIT Press, 1965.

5. Hayes, Curtis W., Jacob Ornstein, and William W. Gage, ABC's of Language and Linguistics, Institute of Modern Languages, Inc., 1977.

6. Fillmore, Charles, "A Case For Case," Universals in Linguistic Theory ed. Bach, Emmon and Robert T. Harms, Holt, Rinehart, and Winston, 1968.

7. Schank, Roger C., "Identifications of Conceptualizations Underlying Natural Language," Computer Models of Thought and Language, ed. Schank, Roger and Kenneth Mark Colby, W.H. Freeman and Company, 1973.

8. Clocksin, W.F. and C.S. Mellish, Programming in Prolog, Springer-Verlay, 1981.

BIBLIOGRAPHY

Bolinger, Dwight, Aspects of Language, Harcourt Brace Jovanovich, Inc, 1975.

Chomsky, Noam, The Logical Structure of Linguistic Theory, Plenum Press, 1977.

Dik, Simon C. ed.,Advances in Functional Grammar, Foris Publications, 1983.

Dik, Simon C., Studies in Functional Grammar, Academic Press, 1980.

Dineen, Francis P., An Introduction to General Linguistics, Holt, Rinehart, and Winston, Inc., 1967.

Gleason, H.A.jr, Linguistics and English Grammar, Holt, Rinehart, and Winston, Inc., 1965.

Hoekstra, Teun, Harry van der Hulst, and Michael Moortgat eds., Perspectives in Functional Grammar, Foris Publications, 1981.

Lyons, John, Introduction to Theoretical Linguistics, Cambridge University Press, 1968.

Schank, Roger, "The Conceptual Analysis of Natural Language," Natural Language Processing, ed. Rustin, Randall, Algorithmics Press, Inc, 1973.

Wardhaugh,Ronald, Introduction to Linguistics, Institute of Modern Languages, McGraw Hill, 1977.

Winston, Patrick Henry, Artificial Intelligence, Addison-Wesley Publishing Company, 1984.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5100	2
3.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943-5100	2
4.	Curricular Officer, Code 37 Computer Technology Naval Postgraduate School Monterey, California 93943-5100	1
5.	Roger G. Marshall Department of Computer Science U.S. Naval Academy Annapolis, Maryland 21402-5002	5
4.	Lcdr Fred G. Orchard USS Joseph Strauss (DDG-16) Fleet Post Office, San Francisco, California 96678-1246	10

2168;6

Thesis
0583429
c.1

Orchard
Implementation of a natural language processor using Functional Grammar.